ISSN 0542-9951

P. 1877/94

8 1994

informatyka

Systemy
czasu rzeczywistego

WYDAWNICTWO SIGMA NOT

Cena 45 000 zł

informatyka

nr 8 1994
Sierpień
Rok wydania XXX

Miesięcznik
ISSN 0542-9951
INDEKS 36124

KOLEGIUM REDAKCYJNE:

mgr Jarosław DEMINET
mgr inż. Piotr FUGLEWICZ
prof. dr hab. Jan GOLIŃSKI
mgr Teresa JABŁOŃSKA
(sekretarz redakcji)
mgr Krystyna
KARWICKA-RYCHLEWICZ
Władysław KLEPACZ
(redaktor naczelny)
mgr inż. Jan RYZKO
dr Witold STANISZKIS
dr hab. Kazimierz SUBIETA
dr Zdzisław SZYJEWSKI

PRZEWODNICZĄCY
RADY PROGRAMOWEJ:

Prof. dr hab.
Juliusz Lech KULIKOWSKI

WYDAWCA:
Wydawnictwo Czasopism i Książek
Technicznych SIGMA NOT
Spółka z o.o.
ul. Ratuszowa 11
00-950 WARSZAWA
skrytka pocztowa 1004

Redakcja:
01-552 Warszawa,
Pl. Inwalidów 10, p. 104, 105
tel., fax: 39-14-34

Materiałów nie zamówionych
redakcja nie zwraca

W sprawach ogłoszeń prosimy zwracać się bezpośrednio do Redakcji lub Działu Reklamy i Marketingu 00-950 Warszawa ul. Mazowiecka 12 telefon: 27-43-66 telefaks: 26-80-16

W numerze:

	Strona
Bez uszczęśliwiania na siłę – wywiad z *Piotrem Fuglewiczem*	1
Niektóre zagadnienia związane z projektowaniem i eksploatacją oprogramowania na poziomie profesjonalnym – *Zdzisław Żurakowski*	4
Analiza bezpieczeństwa komputerowego systemu sterowania – *Janusz Górski, Andrzej Wardziński*	7
Projektowanie oprogramowania systemów czasu rzeczywistego. Podstawowe fazy projektu – *Zbigniew Huzar*	11
Systemy czasu rzeczywistego – metody specyfikacji – *Zbigniew Huzar, Jan Magott*	14
Standardy planowania jakości, dokumentacji i konserwacji oprogramowania – *Krzysztof Sacha*	18
Testowanie oprogramowania – *Elżbieta Hudyma*	22
Metody zbiorów rozmytych w analizie decyzyjnej przedsiębiorstwa. Część 1 – *Tadeusz Witkowski*	26
Polemiki	
Odpowiedź na polemikę „Co to jest uniwersytet?" Andrzeja Horodeńskiego (ENTER nr 6/94) – *Kazimierz Subieta*	31

W najbliższych numerach:

- **Marek Stabrowski prezentuje podprogramy w języku symulacji systemów dynamicznych AMIL**
- **Janusz Stokłosa omawia międzynarodową standaryzację w zakresie technik kryptograficznych.**
- **A.W. Scheer z zespołem prezentuje system ARIS obejmujący metodykę i oprogramowanie dla modelowania i realizacji zintegrowanych systemów informacyjnych.**
- **Adam Mrózek i Ryszard Winiarczyk omawiają istotę oraz przykład systemu operacyjnego czasu rzeczywistego.**
- **Tadeusz Witkowski przedstawia drugą część artykułu na temat metod zbiorów rozmytych w analizie decyzyjnej przedsiębiorstwa.**

Warunki prenumeraty na 1994 r.

Zamówienia na prenumeratę czasopism wydawanych przez Wydawnictwo SIGMA-NOT można składać w dowolnym terminie. Mogą one obejmować dowolny okres czasu, tzn. dotyczyć dowolnej liczby kolejnych zeszytów każdego czasopisma.

Zamawiający może otrzymywać zaprenumerowany przez siebie tytuł począwszy od następnego miesiąca po dokonaniu wpłaty. Zamówienia na zeszyty sprzed daty otrzymania wpłaty będą realizowane w miarę możliwości – z posiadanych zapasów magazynowych.

Warunkiem przyjęcia i realizacji zamówienia jest otrzymanie z banku potwierdzenia dokonania wpłaty przez prenumeratora. Dokument wpłaty jest równoznaczny ze złożeniem zamówienia.

Wpłat na prenumeratę można dokonywać na ogólnie dostępnych blankietach w Urzędach Pocztowych (przekazy pieniężne) lub Bankach (polecenie przelewu), przekazując środki pod adres:
Wydawnictwo SIGMA-NOT Spółka z o.o.
Zakład Kolportażu
00-716 Warszawa, skr. poczt. 1004
konto:
PBK S.A. III 0/Warszawa nr 370015-1573-139-11

Wpłaty na prenumeratę przyjmują także wszystkie urzędy pocztowe nadawczo-odbiorcze oraz doręczyciele na terenie całego kraju

Na blankiecie wpłaty należy czytelnie podać nazwę zamawianego czasopisma, liczbę zamawianych egzemplarzy, okres prenumeraty oraz własny adres.
Na życzenie prenumeratora, zgłoszone np. telefonicznie, Zakład Kolportażu ul. Bartycka 20, 00-950 Warszawa, (telefony: 40-30-86, 40-35-89 oraz 40-00-21 wew. 249, 293, 299) wysyła specjalne blankiety zamówień wraz z aktualną listą tytułów i cennikiem czasopism.

Odbiorcy zagraniczni mogą otrzymywać czasopisma poprzez prenumeratę dewizową (wpłata dokonywana poza granicami Polski w dewizach, wg cennika dewizowego z cenami podanymi w dolarach amerykańskich) lub poprzez zamówioną w kraju prenumeratę ze zleceniem wysyłki za granicę (zamawiający podaje dokładny adres odbiorcy za granicą, dokonując równocześnie wpłaty w wysokości dwukrotnie wyższej niż cena normalnej prenumeraty krajowej).

Egzemplarze archiwalne (sprzedaż przelewowa lub za zaliczeniem pocztowym) można zamawiać pisemnie, kierując zamówienia pod adresem: Wydawnictwo SIGMA NOT, Spółka z o.o. Zakład Kolportażu, 00-716 Warszawa, ul. Bartycka 20, paw. B, tel. 40-37-31, natomiast za gotówkę można je nabyć w Klubie Prasy Technicznej w Warszawie ul. Mazowieckiej 12, tel. 26-80-17.

Istnieje możliwość zaprenumerowania 1 egz. czasopisma po cenie ulgowej przez indywidualnych członków stowarzyszeń naukowo-technicznych zrzeszonych w FSNT oraz przez uczniów zawodowych i studentów szkół wyższych. Blankiet wpłaty na prenumeratę ulgową musi być opatrzony na wszystkich odcinkach pieczęcią koła SNT lub szkoły.

W przypadku zmiany cen w okresie objętym prenumeratą Wydawnictwo zastrzega sobie prawo do wystąpienia o dopłatę różnicy cen oraz prawo do realizowania prenumeraty tylko w pełni opłaconej.
Cena jednego egzemplarza: normalna 45 000 zł, ulgowa 31 500 zł
Wartość prenumeraty w zł:
Normalna: kwartalna 135 000, półroczna 270 000, za trzy kwartały 404 000
Ulgowa: kwartalna 94 500, półroczna 189 000, za trzy kwartały 283 500

Skład i druk: Drukarnia SIGMA NOT Sp. z o.o. z. 265/94

P. 1877/94

Biblioteka Główna Politechniki Śląskiej Gliwice

Bez uszczęśliwiania na siłę

Z Piotrem Fuglewiczem prezesem Polskiego Towarzystwa Informatycznego rozmawia Krystyna Karwicka-Rychlewicz

– Przez wiele lat PTI było jedyną organizacją działającą w środowisku informatyków. Obecnie są cztery. Jak w związku z tym zmienił się zakres działania Towarzystwa?

– PTI przyjęło z dużą radością powstanie innych organizacji, nigdy bowiem nie cierpiało i nadal nie cierpi na kompleks oblężonej twierdzy. Do niedawna musiało zajmować się wieloma sprawami i na zewnątrz reprezentować interesy, nie do końca wspólne, a czasem wręcz sprzeczne, różnych grup swoich członków. Sytuacja, kiedy jasno jest określone, kto się czym zajmuje – jest bardzo dobra. PTI ciągle jest stowarzyszeniem naukowym, staje się coraz bardziej reprezentantem interesów ludzi wykonujących zawody informatyczne. Nie wchodzimy więc sobie w drogę ze Stowarzyszeniem „PRO", które dba o interesy twórców oprogramowania czy PIIiT, która jest organizacją samorządową. Nie moją rolą jest wyznaczanie zakresu działania innym, ale myślę, że gdyby Stowarzyszenie Rozwoju Systemów Otwartych zgromadziło użytkowników informatyki, stając się organizacją konsumencką – wypełniłoby istotną lukę w tym systemie. Konieczna jest bardzo dobra i ścisła współpraca pomiędzy naszymi organizacjami. Tak zresztą jest, tym bardziej, że podstawowe kadry PIIiT to ludzie, którzy wiele lat byli działaczami PTI i nadal są jego członkami. Wycofali się z prac ZG PTI i dlatego, by była jasność poglądów: kto i w czyim imieniu co mówi. Wszystkie nasze organizacje dojrzewają, stają się lepsze i skuteczniejsze w działaniu. PTI zrzekło się jakichkolwiek paternalistycznych ciągot w stosunku do innych stowarzyszeń, którym zresztą oddało swoje najlepsze kadry. I ma swoje jasno określone miejsce.

– Która z tych organizacji liczy najwięcej członków?

– Oczywiście PTI, które ma najdłuższą tradycję; po skreśleniu „martwych dusz" liczy około dwu tysięcy członków i od roku ich z dnia na dzień przybywa.

– Jakie problemy środowiska PTI uważa Pan obecnie za najważniejsze?

– Sprawy zawodowe. I dlatego sądzę, że najważniejszą rzeczą jaką zrobiliśmy, mimo drobnych, przejściowych kłopotów, było podpisanie 3 grudnia ubiegłego roku z British Computer Society porozumienia na temat wprowadzenia w Polsce Systemu Rozwoju Umiejętności, błędnie uważanego za system specjalizacji zawodowej. Tu nie chodzi o specjalizacje, lecz o bardzo dobrze opracowany i stosowany szeroko w Wielkiej Brytanii system, który z jednej strony sprawia, że pracodawca zobligowany jest do troski o rozwój umiejętności zawodowych pracownika, z drugiej zaś umożliwia pracodawcy dużo lepsze zarządzanie personelem. Jest to tzw. *personal management*, czyli wykorzystywanie ludzi zgodnie z ich kwalifikacjami i możliwościami. Ten system jest odpowiedzią na najważniejszą bolączkę zgłaszaną przez członków Towarzystwa: dziki, żywiołowo rozwijający się rynek pracy. PTI wyraźnie przesuwa więc punkt ciężkości swojego działania ze spraw naukowych na zawodowe. Jesteśmy zresztą członkiem międzynarodowej organizacji CEPIS (*Community of European Professional Informatics Societies*), czyli wyraźnie jest mowa o profesji informatyka, a nie o powołaniu czy posłaniu. To przesunięcie nastąpiło jak gdyby samoistnie. Koledzy, którzy zakładali Towarzystwo – jeszcze będąc na uczelni – w dużej części przenieśli się do firm. Niektórzy z nich zbudowali nawet duże własne firmy i będąc kapitalistami, przedsiębiorcami w sferze usług informatycznych, są nadal członkami Towarzystwa. Jednocześnie do PTI zaczęło przychodzić coraz więcej osób wykonujących zawód informatyka, bo uważali, że Towarzystwo może dać im wiedzę i kontakty. Duża grupa kolegów przeszła z nauki do zastosowań informatyki, do wdrożeń. Myślę, iż fakt, że po dwóch wielkich prezesach PTI – profesorach Władysławie M. Turskim i Andrzeju Blikle – ja, praktyk, zostałem prezesem, jest dowodem przesunięcia akcentów w Towarzystwie. To nie znaczy, że zrzekamy się pełnienia roli stowarzyszenia naukowego albo, że w naszych działaniach koledzy z uczelni mają mniejszy priorytet.

– Jak by Pan więc określił generalną linię działania Towarzystwa?

– To nie Towarzystwo określa linię generalną, a później zgodnie z nią działa. Jest dokładnie na odwrót: ci, którzy czują się członkami Towarzystwa, robią coś, co uznają za użyteczne nie tylko dla siebie, ale i dla innych. Towarzystwo jedynie stara się zbierać w jeden nurt te strumyki i mówiąc obrazowo – umacniać wały wokół tej rzeki, stwarzając ludziom możliwości kontaktów, stykania się ze sobą, rozmów, nie usiłując kształtować, a jedynie wzmacniać pozytywne – naszym zdaniem – zjawiska zachodzące wokół nas. Takim „wzmacniaczem" było np. Akademickie Forum Informatyki, które odbywało się w okresie tegorocznych targów INFOSYSTEM. Wiemy , że akademickie środowisko jest zdezintegrowane, na uczelniach o dwie ulice dalej dublowane są pewne tematy, nie ma bowiem między nimi przepływu informacji. Postanowiliśmy zebrać w jednym miejscu ludzi, którzy na uczelniach prowadzą jakieś projekty. Chociaż nie zjawili się przedstawiciele kilku wielkich uczelni, na AFI spotkali się przedstawiciele z 20 uczelni i instytutów. To jest już na tyle dużo, by można było powiedzieć, że nastąpiło zebranie środowiskowe służące lepszemu kontaktowi pomiędzy ludźmi zajmującymi się informatyką nieomal społecznie. Ich dochody są bowiem niskie, ale trzyma ich na uczelniach pasja.

Mówi mgr Bogusław Zalewski – wiceprezes Zarządu Międzynarodowych Targów Poznańskich

Wydaje mi się, że inicjatywa podjęta przez PTI zorganizowania w czasie trwania targów INFOSYSTEM Akademickiego Forum Informatycznego, wsparta przez Komitet Badań Naukowych i MTP jest jak najbardziej na czasie. Otóż od kilku lat oferta nauki polskiej na targach o randze międzynarodowej właściwie nie jest prezentowana lub też prezentowana jest w sposób niezauważalny, niewystarczający. AFI jest próbą zmiany tej sytuacji. Zaproszono 120 osób reprezentujących ośrodki naukowe, 27 skorzystało. Ta próba się udała, można będzie ją ponowić na przyszłorocznym INFOSYSTEMIE, a być może i na innego typu targach. Ta forma promowania nauki jest dość zgrabna i finansowo strawna dla uczelni i instytutów naukowych. Organizatorzy nie traktują tego komercyjnie, w organizację AFI włożono dużo pracy społecznej. To pozwoli skutecznie promować naukę wśród praktyków odwiedzających Targi. Wiele zależy od postawy naukowców, od ich odzewu na tę propozycję. Akademickie Forum Informatyczne, mam nadzieję, to dobry początek promocji nauki.

– Powszechnie wiadomo, że wielu polskich informatyków pracujących na uczelniach wyjechało za granicę, dużo odeszło do firm. A w ogóle informatyka tak naprawdę rozwija się w laboratoriach wielkich firm, uczelniom została dydaktyka.

– To prawda, że uczelnie bywają jedynie dostawcą podstawowych badań, natomiast rozwój technik informatycznych odbywa się za pieniądze wielkich korporacji, których stać na najlepszych. Ale to co czytamy czasem w prasie, że najlepsi polscy informatycy wyjechali, a w kraju zostali najmniej aktywni i właściwie to nie mamy żadnej nauki, nie jest prawdą. Nie sądzę, by sytuacja w polskiej nauce informatycznej była znacznie gorsza niż w innych krajach, chociażby w Niemczech, Francji czy Wielkiej Brytanii. Naszym kolegom z uczelni wmawia się kompleksy, mówiąc: jesteście brzydcy, źli i w ogóle nikt was nie kocha. Tymczasem oni są tacy sami, jak ich koledzy na całym świecie. Zresztą oni to wiedzą, bo wszyscy prawie się o ten świat otarli. By zaistniał jeden Einstein, potrzebnych było kilka tysięcy fizyków, którzy nie mieli powszechnie znanych osiągnięć, ich publikacje gdzieś tam się ukazały w kilkudziesięciu egzemplarzach. W informatyce jest na świecie kilkanaście wielkich nazwisk, na każde z nich pracuje, pośrednio lub bezpośrednio, kilkuset ludzi. Mam nadzieję, że Towarzystwo zintegruje nasze środowisko i poprawi jego samopoczucie. To ważne, by wierzyli, że to co robią ma sens, by doskonalili swoją wiedzę, od kadry uczelnianej zależy bowiem wiedza i jakość następców.

– Jak Pan ocenia najmłodszą kadrę informatyczną?

– Polskie uczelnie dobrze przygotowują teoretycznie studentów, niestety – mówię to z pełną odpowiedzialnością jako praktyk, który ich zatrudnia – absolwenci nie są wystarczająco przygotowani praktycznie do wykonywania zawodu. Brak im też świadomości, iż informatyka jest służebnym rzemiosłem społecznym. Całą tę pracę wykonujemy przecież po to, by jakiś człowiek, który może mieć zielone pojęcie o informatyce, mógł wykonywać swoją pracę w sposób wygodniejszy, łatwiejszy i szybszy. Tymczasem w wielu uczelniach uczy się po to języka C, by mieć radość z jego używania, a nie po to, by służył on jako narzędzie. To tak, jak gdyby na budownictwie czerpać rozkosz z nauki o kielni.

– Bardzo się cieszę, że Pan to powiedział. Jeszcze dziś bowiem wielu informatyków z pewną jakby wyższością traktuje użytkowników. Dla naszych komputerowych guru człowiek, który zna jedynie edytor tekstów czy arkusz kalkulacyjny, bo tylko ta wiedza jest mu potrzebna, to profan, któremu nie warto poświęcić nawet chwili uwagi...

– To dziecięca choroba informatyki, z której zaczynamy powoli wyrastać. Nie mogę jednak nie wrzucić kilku kamyczków do ogródka użytkowników. Otóż nie kupuje się samochodu nie mając prawa jazdy lub szofera. Natomiast często kupuje się komputer, a nawet cały system informatyczny, nie wiedząc po co on właściwie jest i do czego służy. To sprawa edukacji informatycznej społeczeństwa. A także kształcenia analityków systemów, którzy pomagali użytkownikom sformułować ich potrzeby. Próbuje się ich kształcić, ale niestety, koledzy na uczelniach mają na ten temat wiedzę teoretyczną. Prawdziwej wiedzy o tym, jak buduje się systemy informatyczne, nabywa się w zetknięciu z praktyką. Tymczasem uczelnie niechętnie sięgają po ludzi związanych z praktyką, z drugiej strony ci ludzie są dość zajęci. W nawiązywaniu kontaktów teoretyków z praktykami, w wymianie wzajemnych doświadczeń i wiedzy całego środowiska, widzę rolę PTI. Myślę, że o tych wszystkich sprawach będziemy rozmawiać na Kongresie Informatyki Polskiej, który ma się odbyć pod koniec tego roku. PTI jest jednym z czterech sygnatariuszy jego programu przygotowawczego.

– Czy PTI podnosi prestiż środowiska decydentów i w społeczeństwie?

– Już kiedyś ktoś cynicznie powiedział: pomagać trzeba tym, którzy sami potrafią sobie pomóc; pomaganie bezradnym to strata czasu. Towarzystwo jest, lub chce być, czymś w rodzaju wzmacniacza wszystkich pozytywnych inicjatyw środowiska. Musimy mieć jednak świadomość, że PTI to nie prezes, to nie zarząd, ale ludzie w Lublinie, Zielonej Górze czy Warszawie. Rola PTI rośnie przez aktywność jego członków. Chcemy umożliwiać kontakty, integrować środowisko, wzmacniać jego głos, nie zaś uszczęśliwiać je na siłę.

– Dziękujemy Panu za rozmowę.

Kongres IFIP'94

Już wkrótce, bo w dniach od 28 sierpnia do 2 września odbędzie się w Hamburgu (RFN) kolejny 13. Światowy Kongres IFIP (*International Federation for Information Processing*), niewątpliwie największa rozmiarem i zasięgiem oraz długością tradycji międzynarodowa konferencja informatyków. Kongresy IFIP były regularnie organizowane od 1959 r. co 3 lata, natomiast podczas ostatniego Kongresu w 1992 r. w związku z przyspieszeniem tempa rozwoju technologii informatycznej cykl ten skrócono na dwuletni. IFIP jest federacją organizacji informatyków (lub ich krajowych zrzeszeń), skupiającą obecnie 43 takich organizacji, reprezentujących 53 kraje. Składa się z 11 Komitetów Technicznych (*Technical Committees*), obejmujących znaczną liczbę wyspecjalizowanych grup roboczych (*Working Groups*), pokrywających praktycznie cały obszar naukowych i praktycznych zainteresowań informatyków. Stąd Kongresy IFIP stanowią najbardziej wszechstronne i kompletne odzwierciedlenie aktualnego stanu i perspektyw światowego rozwoju informatyki, a ostatnio również telekomunikacji, oraz problemów nurtujących działające w tych dziedzinach środowisko.

Gospodarzem, a jednocześnie współorganizatorem tegorocznego Kongresu jest niemieckie Towarzystwo Informatyczne GI (*Gesellschaft für Informatik*), świętujące w tym roku 25-lecie działalności. Jego celami, podobnie jak innych ogólnokrajowych stowarzyszeń informatycznych, są wspomaganie działalności badawczo-rozwojowej, rozwoju zastosowań oraz kształcenia zawodowego. GI zrzesza obecnie ok. 20 000 członków oraz obejmuje ponad 120 wyspecjalizowanych komitetów technicznych i grup roboczych, ściśle współpracujących z ich odpowiednikami w IFIP. Towarzystwo organizuje rocznie kilkadziesiąt krajowych i międzynarodowych imprez oraz prowadzi szeroką działalność wydawniczą (w tym również wydaje kilka czasopism).

Miejscem obrad tegorocznego Kongresu jest nowoczesne Centrum Kongresowe w Hamburgu, a wiodącym hasłem „Ewolucja komputera i telekomunikacji – siłą napędową rozwoju" („*Computer and Communications Evolution – The Driving Forces*"). O stronie merytorycznej Kongresu, a zwłaszcza selekcji nadsyłanych na Kongres materiałów decydował jego 42-osobowy Komitet Programowy, w skład którego wchodził również przedstawiciel Polski (L. Pacholski z Uniwersytetu Wrocławskiego). Program Konferencji będzie niezwykle obszerny (nadesłany informator zawiera ponad 100 stron formatu A_5 zapisanego drobną czcionką) i obejmuje wyjątkowo szeroki i klarownie pogrupowany tematycznie, wachlarz problemów. W ciągu 5 dni na sesjach przed i popołudniowych od 9^{00} do 19^{00} zostanie wygłoszonych przez reprezentantów ok. 20 krajów ponad 120 referatów.

Podobnie jak to kształtuje się w rozwoju technologii informatycznej, największą grupę autorów stanowią przedstawiciele USA, Japonii i gospodarzy. Kraj nasz reprezentuje niestety tylko jeden autor, chociaż nazwiska polskie, w tym także osób znanych uprzednio z działalności w kraju, pojawiają się w programie wielokrotnie. Tak skromna, w porównaniu do możliwości, reprezentacja Polski wypada szczególnie żenująco na tle zapowiedzianych wystąpień tak niewielkich krajów, jak kraje bałtyckie, Chorwacja czy Słowacja.

Szczególnie dużą część obrad stanowią dyskusje panelowe dające uczestnikom możliwość bardziej aktywnego udziału w przedyskutowaniu najbardziej istotnych problemów współczesnej informatyki. Konstrukcja programu Kongresu jest bardzo przejrzysta przez wyodrębnienie pięciu podstawowych ciągów tematycznych (ang. *tracks*), którymi są: technologia sprzętu i oprogramowania, zastosowania komputerów i telekomunikacji, wpływ komputerów i telekomunikacji na rozwiązaniej głównych problemów współczesnego świata, podstawy naukowe informatyki, informatyka w krajach rozwijających się. W ramach powyższych głównych ciągów przeprowadzono bardziej szczegółowy podział tematyki, umożliwiający każdemu uczestnikowi Kongresu wybór referatów oraz dyskusji panelowych zbieżnych z osobistymi zainteresowaniami. Dla potencjalnego uczestnika z Polski istotnym walorem tegorocznego Kongresu jest jego bliskość granic Polski (ok. 700 km), natomiast czynnikiem zniechęcającym, zwłaszcza z kręgów krajowego środowiska naukowego – bardzo wysoki koszt uczestnictwa (990 DM). Powyższe koszty zapewne spowodują, że polskich uczestników nie będzie na IFIP'94 więcej niż na imprezach o znacznie mniej korzystnej dla nas lokalizacji. (*WK*)

Oprogramowanie komputerowych systemów czasu rzeczywistego – zagadnienia związane z wytwarzaniem oprogramowania do zastosowań przemysłowych

Ogólnopolska Konferencja Naukowo-Techniczna Wrocław, 22–23 września 1994 r.

Zamiar zorganizowania Konferencji powstał w związku z istniejącą potrzebą zastosowania tego typu systemów w kolejnictwie, a ściślej w taborze kolejowym, z którym część organizatorów związana jest zawodowo. Jednak ze względu na to, że zakres prac prowadzonych w kraju w tej dziedzinie, nie tylko w kolejnictwie, ale – sądząc na podstawie publikacji – w ogóle, jest badzo niewielki, a dobre praktyki projektowania oprogramowania dla systemów czasu rzeczywistego są w decydującym stopniu wspólne dla wszystkich zastosowań, postanowiono objąć zakresem konferencji wszystkie dziedziny, w których takie systemy są stosowane. W przypadku Polski oznacza to głównie:

- kolejnictwo (sterowanie ruchem kolejowym),
- elektroenergetykę,
- sterowanie ruchem lotniczym,
- przemysł taboru kolejowego,
- przemysł lotniczy
- robotykę i automatyczne sterowanie procesami technologicznymi,
- medycynę, w tym systemy nadzoru bezpośredniego,
- systemy przeznaczone dla wojska.

W początkach zastosowania komputerów do sterowania w czasie rzeczywistym, a więc w latach sześćdziesiątych, główne ograniczenia wynikały z części sprzętowej takich systemów (która była bardzo zawodna), co ze zrozumiałych względów odgrywa mniejszą rolę w zastosowaniach komputerów do obliczeń.

Sytuacja zmieniła się radykalnie w latach siedemdziesiątych, kiedy to – po wynalezieniu mikroprocesora scalonego – zaczęto na dużą skalę produkować mikrokomputery. Koszt części sprzętowej spadł wówczas gwałtownie, a niezawodność wzrosła o wiele rzędów wielkości. Obecnie część sprzętowa systemów komputerowych jest coraz tańsza i coraz lepsza, a sprzęt ten może być produkowany masowo, osiągając niższe ceny i wyższą jakość.

Główny ciężar problemów, wiążących się z zastosowaniem systemów komputerowych do sterowania w czasie rzeczywistym, przesunął się obecnie na oprogramowanie, które musi być indywidualnie projektowane dla każdego zastosowania. Głównym problemem jest jego zawodność, wynikająca z błędów wprowadzonych na wszystkich etapach jego realizacji.

Osiągnięcie wysokiej jakości oprogramowania tego typu jest bardzo trudnym zadaniem o charakterze technicznym i organizacyjnym, które nawet w krajach przodujących nie jest zadowalająco rozwiązane. Na przykład, w samej tylko Wielkiej Brytanii z kwot, które przemysł, rząd i handel przeznaczyły na oprogramowanie, co najmniej 2 mld funtów rocznie było marnowanych wskutek złej jakości oprogramowania.

Konferencja będzie poświęcona głównie zagadnieniom związanym z praktyką inżynierską wytwarzania oprogramowania dla komputerowych systemów czasu rzeczywistego stosowanych na skalę przemysłową, a więc np. jako wbudowane systemy sterowania i diagnostyki w seryjnie produkowanych pojazdach trakcyjnych.

Zakres tematyczny konferencji został podany w obszernej informacji, zamieszczonej w numerze październikowym INFORMATYKI z 1993 r.

Spośród prezentowanych na konferencji wybrano do publikacji w obecnym numerze referaty o bardziej ogólnym charakterze, dotyczące:

- niektórych zagadnień prawnych, normalizacyjnych, organizacyjnych, atestacji oraz innych, związanych z projektowaniem i eksploatacją oprogramowania na profesjonalnym poziomie,
- analizy bezpieczeństwa systemów komputerowych,
- definicji podstawowych faz cyklu projektowania oprogramowania,
- metod specyfikacji wymagań,
- przeglądu norm ANSI/IEEE określających zasady: planowania i kontroli jakości oprogramowania, dokumentowania projektów, organizacji procesu pielęgnacji oprogramowania oraz atestacji wprowadzanych zmian,
- testowania oprogramowania.

Wykaz pozostałych referatów, opublikowanych w materiałach konferencyjnych, zamieszczono wewnątrz numeru. Najciekawsze z nich zostaną opublikowane we wkładce pn. SYSTEMY CZASU RZECZYWISTEGO, której pierwszy numer zaplanowano w następnym numerze INFORMATYKI.

W trakcie organizowania Konferencji okazało się, że jest to prawdopodobnie pierwsza w Polsce konferencja na temat projektowania oprogramowania dla zastosowań przemysłowych. Zapóźnienie Polski w tej dziedzinie jest tak duże, że wiele terminów angielskich nie ma w języku polskim swoich dobrze zakorzenionych i jednoznacznie ustalonych odpowiedników. Dlatego nieraz nawet po starannym przetłumaczeniu danego fragmentu, w języku polskim brzmi on trochę obco. Jest to – niestety – również powodem wielu rozbieżności terminologicznych w artykułach konferencyjnych, w tym także publikowanych w niniejszym numerze INFORMATYKI. Uregulowanie tak ważnej sprawy będzie wymagało świadomych działań w tym kierunku.

Ogólnie na konferencję przyjęto 30 referatów, 3 referaty zostały zamówione. Zgłoszony został udział 76 uczestników z 30 różnych instytucji cywilnych i wojskowych: przedsiębiorstw państwowych, spółek z udziałem kapitału zagranicznego, firm prywatnych, resortowych placówek naukowych i projektowych oraz szkół wyższych.

Organizatorami konferencji są:

- Sekcja Trakcji Elektrycznej Oddziału Wrocławskiego Stowarzyszenia Elektryków Polskich,
- Dolnośląski Oddział Polskiego Towarzystwa Informatycznego,
- Centrum Informatyczne Politechniki Wrocławskiej,
- Fabryka Wagonów PAFAWAG we Wrocławiu.

W skład Komitetu Programowego konferencji wchodzą:

- **Przewodniczący – prof. dr hab. inż. Zbigniew HUZAR,** Kierownik Centrum Informatycznego Politechniki Wrocławskiej.
- **Wiceprzewodniczący – mgr inż. Zdzisław ŻURAKOWSKI,** kierownik Samodzielnej Pracowni Pojazdowych Systemów Komputerowych przy F.W. PAFAWAG, członek Kolegium Sekcji Trakcji Elektrycznej Oddziału Wrocławskiego SEP, prezes koła SEP przy F.W. PAFAWAG;
- **prof. dr hab. inż. Jan MAGOTT** z Instytutu Cybernetyki Technicznej Politechniki Wrocławskiej;
- **prof. dr hab. Zygmunt MAZUR** z Centrum Informatycznego Politechniki Wrocławskiej, wiceprezes Oddziału Dolnośląskiego PTI;
- **dr inż. Ireneusz JÓŹWIAK** z Centrum Informatycznego Politechniki Wrocławskiej, prezes Oddziału Dolnośląskiego PTI.

Konferencja odbędzie się w Fabryce Wagonów PAFAWAG **we Wrocławiu, w dniach 22–23 września 1994 r.**

Niektóre zagadnienia związane z projektowaniem i eksploatacją oprogramowania na poziomie profesjonalnym

Zdzisław Żurakowski
F.W. PAFAWAG

Trwający od kilkunastu lat szybki wzrost zastosowań systemów komputerowych czasu rzeczywistego, w tym także systemów wbudowanych w urządzenia lub obiekty techniczne, spowodował, że systemy te już obecnie w krajach wysoko rozwiniętych odgrywają znaczącą rolę w zapewnieniu wysokiego poziomu technicznego i użytkowego wielu urządzeń i obiektów technicznych, np. taboru kolejowego, i rola ta będzie nadal rosła. Ponieważ część sprzętowa takich systemów może opierać się w znacznym stopniu na wysokiej jakości urządzeniach seryjnych, a więc stosunkowo tanich i łatwo dostępnych, w czasopismach zagranicznych wyrażany jest pogląd, że w przyszłości oprogramowanie, które musi być programowane indywidualnie do każdego zastosowania i które decyduje o poziomie rozwiązania, będzie miało krytyczne znaczenie dla osiągnięcia wysokiego poziomu większości budowanych urządzeń i obiektów technicznych.

Projektowanie oprogramowania dla tego typu zastosowań przestało już być wynikiem prac naukowo-badawczych oraz indywidualnego, w znacznym stopniu nieformalnego podejścia (czymś, co robiło wrażenie działalności trochę artystycznej, jak pisano w jednej z publikacji zachodnich), a nabrało – i nabiera w dalszym ciągu – cech normalnego podejścia inżynierskiego, jakiemu podlegają wszystkie inne dziedziny produkcji przemysłowej (nie przestając być w dużym stopniu aktem twórczym w sferze koncepcji). Ponadto, rozpowszechnianie się systemów czasu rzeczywistego, spełniających coraz bardziej złożone i odpowiedzialne funkcje, wiąże się ze wzrostem złożoności oprogramowania i stawianych mu jakościowych wymagań.

Projektowanie

Z publikacji zagranicznych wynika jednoznacznie, że dla dużego systemu czasu rzeczywistego nie jest możliwe opracowanie oprogramowania na dobrym, profesjonalnym poziomie, w ten sposób, że siada 3–4 programistów i pisze programy, które następnie łączy się i uruchamia.

Jeśli taki połączony program będzie nawet działał, to na ogół trudno coś powiedzieć o jego niezawodności, bezpieczeństwie, jakości itp. Trudno powiedzieć, ile i jak, trzeba go testować, aby uzyskać wystarczającą pewność niezawodnej realizacji założonych funkcji. Nie wiadomo, jak dokonać atestacji takiego oprogramowania, skoro proces projektowania nie był zorganizowany według określonych zasad, np. zgodnie z obowiązującą normą, i zwykle nie ma odpowiednich dokumentów z poszczególnych faz projektu.

Mgr inż. ZDZISŁAW ŻURAKOWSKI ukończył w 1967 r. Wydział Elektryczny Politechniki Wrocławskiej (specjalność: elektroenergetyczna automatyka zabezpieczeniowa). Pracował w Instytucie Energetyki w Warszawie, Instytucie Energoelektryki Politechniki Wrocławskiej oraz w Hucie Siechnice k. Wrocławia. Od 1987 r. pracuje w OBR Budowy Taboru Kolejowego we Wrocławiu, (obecnie F.W. „PAFAWAG") aktualnie na stanowisku kierownika Samodzielnej Pracowni Pojazdowych Systemów Komputerowych. Jest autorem licznych publikacji i 2 patentów, głównie z zakresu elektroenergetycznej automatyki zabezpieczeniowej oraz zastosowania systemów komputerowych w taborze kolejowym.

W tym świetle wszystkie programy, które powstały w podobny sposób, można traktować jedynie jako prace studialne w zakresie algorytmów, lub koncepcji takich systemów. Nie są to programy do praktycznych realizacji, zwłaszcza dla systemów dużych, którym stawia się wysokie wymagania w zakresie dyspozycyjności, niezawodności lub bezpieczeństwa.

Podejmując w Polsce próbę wykonania projektu oprogramowania dla komputerowego systemu czasu rzeczywistego przeznaczonego do zastosowań przemysłowych na profesjonalnym poziomie, napotyka się obecnie na poważną barierę w postaci zupełnego braku wzorców dla projektów tego typu. Brak jest w tym zakresie polskich norm i uregulowań prawnych, bardzo niska świadomość zagadnień, wśród szerokich kręgów kadry kierowniczej i inżynierskiej pracującej w przemyśle, brak – o ile wiem – laboratoriów zdolnych do oceny bezpieczeństwa i atestacji tego typu systemów (w każdym razie w sektorze cywilnym).

Ponadto istnieje bardzo mało publikacji w języku polskim w czasopismach i w formie książkowej, które umożliwiałyby poznanie problematyki występującej przy projektowaniu oprogramowania dla takich systemów oraz w trakcie jego eksploatacji.

W rocznikach czasopisma INFORMATYKA z lat 1984–1992 zamieszczonych było jedynie kilka artykułów poświęconych temu zagadnieniu, z których tylko jeden, na podstawie norm zagranicznych i międzynarodowych, omawiał powyższe zagadnienia w aspekcie praktycznym i na aktualnym poziomie.

W podręcznikach akademickich polskich autorów [4, 5, 6], omawiających komputerowe systemy czasu rzeczywistego, jeśli nawet są informacje dotyczące oprogramowania takich systemów, to nie występuje w nich w sposób jawny metodologia jego projektowania i testowania, czy atestacji systemów. W wymienionych wyżej podręcznikach w ogóle nie występują takie pojęcia, jak: **specyfikacja wymagań dla systemu** (ang. *system requirements specification*), czy **specyfikacja wymagań dla oprogramowania** (ang. *software requirements specification*), z których drugie jest jednym z podstawowych pojęć inżynierii oprogramowania od chwili jej powstania oraz podstawową fazą projektu oprogramowania, a pierwsze – projektu systemu.

W podręcznikach tych zagadnienia niezawodności systemów komputerowych przedstawione są w minimalnym stopniu i w przestarzały sposób. Nie jest omówiona np. zasada tolerowania defektów, bez której nie można dzisiaj zaprojektować systemu o dużej niezawodności, co często oznacza po prostu użytecznego. Nie są również omówione zasady stosowane w systemach krytycznych ze względu na bezpieczeństwo, ani nie jest nawet analizowane samo pojęcie systemu bezpiecznego. Spotykana w artykułach publikowanych w czasopismach krajowych zasada *fail-safe* (na ogół tylko w postaci nazwy) nie jest w tych podręcznikach wymieniona nawet z nazwy.

Istnieje również niewielka liczba wydanych tłumaczeń książek z tego zakresu. Są to jednak tłumaczenia opracowań wydanych w oryginale w połowie lat siedemdziesiątych i dlatego w znacznym stopniu już przestarzałych.

Atestacja

Zastosowania systemów komputerowych czasu rzeczywistego w takich dziedzinach, jak:

- sterowanie ruchem kolejowym,
- elektroenergetyka,
- sterowanie ruchem lotniczym,

- przemysł taboru kolejowego,
- przemysł lotniczy,
- robotyka i automatyczne sterowanie procesami technologicznymi,
- medycyna, w tym systemy nadzoru bezpośredniego,
- technika wojskowa,

mają przeważnie charakter krytyczny ze względu na bezpieczeństwo osób, mienia lub środowiska. Istotnym zagadnieniem jest wówczas atestacja takich systemów, przed dopuszczeniem ich do produkcji i eksploatacji, oraz nadzór nad nimi w czasie eksploatacji.

Praktyka krajów przodujących w tej dziedzinie dowiodła, że obecnie, pomijając bardzo proste przypadki, nie jest możliwe wytworzenie oprogramowania, które nie zawierałoby błędów. Wiadomo także, że nie jest możliwe takie przetestowanie układu, które wykluczałoby istnienie błędu, oraz że błąd w oprogramowaniu może ujawnić się nawet po kilku latach prawidłowego funkcjonowania systemu.

Również analiza poprawności tego typu programów, zawierających zwykle kilkanaście, kilkadziesiąt lub więcej tysięcy linii kodu, przez zespoły ludzkie nie jest zadaniem łatwym.

Wszystko to powoduje, że atestacja takich systemów pod względem bezpieczeństwa i wydanie certyfikatu zezwalającego na ich stosowanie jest na obecnym etapie rozwoju zadaniem czasochłonnym i bardzo złożonym. Odróżnia to w sposób zasadniczy tego typu systemy od urządzeń podległych Dozorowi Technicznemu, w których ocenę bezpieczeństwa urządzenia dokonuje się na drodze stosunkowo prostych i nielicznych pomiarów, dających jednoznaczne wyniki, oraz oceny stopnia przestrzegania określonych zasad, również bardzo prostych i łatwych do zweryfikowania.

Skutki pojawienia się błędu w oprogramowaniu mogą być bardzo poważne. Z publikacji zachodnich (w tym również podręcznikowych) wynika, że informacje na ten temat są ujawniane na Zachodzie bardzo niechętnie. Jednak nawet z tych, które są ujawniane wynika, że wypadków spowodowanych istnieniem błędów w oprogramowaniu – w tym powodujących ofiary śmiertelne – jest dość dużo i stan taki trwa od wielu lat.

W Polsce prace nad zastosowaniem komputerów do sterowania w czasie rzeczywistym rozpoczęły się bardzo wcześnie, bo jeszcze pod koniec lat pięćdziesiątych. Jednak do dnia dzisiejszego nie wyszły one poza stadium opracowań jednostkowych o charakterze prac naukowo-badawczych. Tego typu systemy nie podlegają Dozorowi Technicznemu. Również ustawa o badaniach i certyfikacji z 3 kwietnia 1993 r. zwalnia od obowiązku certyfikacji tego typu wyroby (art. 13.4) stawiając jedynie warunek, żeby spełnione były wymagania dotyczące bezpieczeństwa pracy i użytkowania oraz ochrony życia, zdrowia i środowiska.

Dopóki istnieje dotychczasowy stan, w którym stosowane są wyłącznie rozwiązania jednostkowe, będące najczęściej wynikiem prac naukowo-badawczych, nie stwarza to – jak wynika z praktyki – żadnych problemów związanych z atestacją i eksploatacją tego typu systemów. W miarę jednak zbliżania się Polski do warunków obowiązujących w krajach rozwiniętych oraz zwiększania się liczby tego typu systemów i wyposażenia w nie obiektów lub urządzeń technicznych produkowanych przez nasz przemysł, należy spodziewać się, że sytuacja będzie wymagała uregulowania, zarówno ze względu na wymagania wewnętrzne, jak i konieczność dostosowania naszego kraju do reszty świata. Na przykład, w krajach Unii Zachodnioeuropejskiej tworzony jest od kilku lat Europejski System Certyfikacji Technologii Informatycznych [2, 3].

Zagadnienia do rozwiązania

Na podstawie publikacji przedstawiających doświadczenia krajów przodujących w tej dziedzinie oraz doświadczeń własnych uważam, że rozwiązanie powinny znaleźć następujące zagadnienia:

- Nadrobienie wieloletnich zaległości w wydawaniu norm z dziedziny oprogramowania dla systemów czasu rzeczywistego, przez wydanie w możliwie krótkim czasie podstawowych norm w tym zakresie oraz powołanie w Polskim Komitecie Normalizacyjnym odpowiedniej Komisji Problemowej, w celu systematycznej działalności normalizacyjnej.
- Uregulowanie prawne wymagań w stosunku do systemów pełniących funkcje krytyczne ze względu na bezpieczeństwo (dopuszczanie do eksploatacji, nadzór w trakcie eksploatacji, rejestracja awarii itd.).
- Opracowanie zunifikowanych metod atestacji i oceny oprogramowania dla takich systemów pod względem (zależnie od potrzeb):
  - bezpieczeństwa
  - zgodności ze specyfikacją wymagań lub z odpowiednią normą,
  - jakości.
- Powierzenie jednej z istniejących placówek naukowych obowiązków związanych z atestacją i oceną oprogramowania lub całych systemów lub powołanie w tym celu nowej placówki.
- Zwiększenie liczby wydawnictw dotyczących rozpatrywanych zagadnień.
- Opracowanie polityki państwa w omawianym zakresie, obejmującej m.in.:
  - finansowanie badań naukowych i innych prac;
  - kształcenie i dokształcanie kadr dla projektowania, nadzoru w eksploatacji oraz kadr kierowniczych;
  - współpracę z organizacjami zagranicznymi, zarówno normalizacyjnymi (jak IEC, ISO), jak i wypracowującymi pewną filozofię np. w dziedzinie wymagań dla systemów krytycznych ze względu na bezpieczeństwo (jak np. EWICS);
  - ustalenie zakresu prac tzw. podwójnego zastosowania w sektorze wojskowym i cywilnym (w literaturze amerykańskiej: *dual-use technology, dual-use software policy* [1]) w celu zwiększenia efektów uzyskiwanych z nakładów na badania, wyposażenie, prace projektowe itp. dla obu tych sektorów;
  - przekształcenie obecnego zaplecza projektowego w dziedzinie oprogramowania, złożonego z nielicznych, rozproszonych zespołów pracujących metodami chałupniczymi, w przemysł oprogramowania.

Ponieważ na każdym niemal kroku odnieść można wrażenie, że szerokie kręgi kadry inżynieryjnej i kierowniczej, pracującej w naszym przemyśle, nie zdają sobie w pełni sprawy z ogromu naszego zapóźnienia w tej dziedzinie i ze zmiany sposobu patrzenia na wiele spraw na świecie, należy im uświadomić, że przemysł oprogramowania to ogromny – obecnie jeden z najbardziej lukratywnych i o kluczowym znaczeniu – dział gospodarki. Przemysł ten np. w samych tylko Stanach Zjednoczonych wytwarza dochód o wartości ponad 100 mld. dolarów rocznie i przewiduje się, że do 2000 r. będie wymagał zatrudnienia dodatkowo 400 tys. profesjonalnie przygotowanych specjalistów [1]. Jest on obecnie intensywnie rozwijany również w Europie Zachodniej, Japonii oraz w niektórych krajach rozwijających się.

Stwierdzenie zawarte w punkcie dotyczącym listy wydawnictw odnosi się w zasadzie do wydawnictw książkowych i innych podobnego typu. Należy jednak stwierdzić, że sytuacja w dziedzinie czasopism również nie nadąża za potrzebami. Jedynym czasopismem, które czasem publikuje poruszane wyżej tematy jest INFORMATYKA. Jednak czasopismo to zajmuje się raczej zagadnieniami ogólnymi, zastosowaniem komputerów w zarządzaniu itp., a publikacje na temat praktycznych aspektów wytwarzania oprogramowania dla systemów czasu rzeczywistego pojawiają się – jak już wspomniano – raczej rzadko. Dlatego pożądane, a nawet niezbędne, byłoby pojawienie się na krajowym rynku wydawniczym odpowiedniego czasopisma specjalistycznego.

* * *

Analiza powyższych zagadnień i wielu dodatkowych, które należałoby uwzględnić dla zintensyfikowania oprogramowania, oraz obecne rozproszenie kompetencji na szereg agend rządowych nasuwają myśl, że najlepszym rozwiązaniem byłoby być może powołanie – wzorem krajów przodujących w tej dziedzinie – zupełnie nowej, centralnej jednostki w randze instytutu, pod nazwą np. Centrum Oprogramowania, która objęłaby zakresem swojego działania wszystkie wyżej wymienione zagadnienia oraz dodatkowo takie zadania, jak:

- koordynację prac prowadzonych w tej dziedzinie przez inne ośrodki;
- współpracę w rozpatrywanym zakresie zagadnień z innymi instytucjami centralnymi i agendami rządowymi (Komitetem Badań Naukowych, Polskim Komitetem Normalizacyjnym, Polskim Centrum Badań i Certyfikacji, zainteresowanymi ministerstwami);
- publikowanie w celach szkoleniowych krytycznych analiz wybranych projektów, od fazy specyfikacji wymagań do fazy wdrożenia do normalnej eksploatacji;
- prowadzenie niektórych badań naukowych, zwłaszcza badań o dużym znaczeniu dla praktyki projektowej;
- prowadzenie prac projektowych, m.in. w celu zdobycia głębokiej i rzetelnej wiedzy na temat problemów występujących w praktyce inżynierskiej. Szczególnie ważne byłoby wykonywanie projektów oprogramowania dla systemów o wysokim stopniu wymagań ze względu na bezpieczeństwo, np. dla kolejnictwa lub dla przemysłu taboru kolejowego, które to zastosowania uznać można za najbardziej krytyczne pod tym względem z zastosowań, jakie mogą obecnie wystąpić w Polsce.

Oprócz zasadniczych funkcji Centrum powinno w miarę potrzeb pełnić również funkcje usługowe, jako centralne laboratorium atestacji oprogramowania, także dla innych systemów niż systemy czasu rzeczywistego.

Jest oczywiste, że aby tego typu placówka mogła odegrać istotną rolę, musiałaby reprezentować odpowiednio wysoki poziom.

Przedstawiając powyższą koncepcję, nie traktuję jej jako konkretnej propozycji, lecz pewną ideę rozwiązania, jedną z możliwych, stosowaną już w krajach przodujących, i którą być może byłoby korzystnie zastosować również w Polsce.

Zważywszy na rolę, jaką komputerowe systemy czasu rzeczywistego odgrywają w rozwoju cywilizacji, należy liczyć się z tym, że utrzymanie stanu takiego jak obecnie, miałoby bardzo dotkliwe konsekwencje nie tylko dla naszego przemysłu, ale dla rozwoju cywilizacji w Polsce w ogóle.

Mając powyższe na względzie, a także interes środowiska zawodowego związanego z tą dziedziną, Komitet Organizacyjny Konferencji skierował listy do osób reprezentujących instytucje centralne, w których kompetencji – według aktualnego Informatora Nauki Polskiej – znajdują się poruszone wyżej sprawy, z prośbą m.in. o przedstawienie w formie krótkiego referatu stanowiska do powyższych zagadnień.

Do chwili przekazania materiałów do publikacji w INFORMATYCE, nie otrzymaliśmy niestety żadnych odpowiedzi.

Powyższy fakt oraz zmiany ustrojowe zachodzące w Polsce oznaczają być może, że instytucje rządowe w coraz mniejszym stopniu będą bezpośrednio ingerowały w kształtowanie naturalnych procesów rozwojowych. Dlatego załatwienie powyższych spraw będzie z pewnością wymagało szybkiej integracji środowiska zawodowego i instytucji zainteresowanych tą problematyką oraz podjęciem odpowiednich inicjatyw.

LITERATURA

[1] Catalanotto K.J.: Computer Software: Federal Policy for a Critical Technology. IEEE Technology and Society Magazine, Winter 1993, pp. 7–12

[2] Gloee G., Westhaeusser R.: Assessment and licensing of control computers in the FRG. Hardware and Software for Real-Time Process Control. Proceedings of the IFIP (IFAC) EWICS Working Conference, Warsaw, Poland, 30 May – 1 June 1988, ELSEVIER Science Publishers, North-Holland, 1989, pp. 555–563

[3] Malagardis N.: Certification and Validation in Europe. Hardware and Software for Real-Time Process Control. Proceedings of the IFIP (IFAC) EWICS Working Conference, Warsaw, Poland, 30 May – 1 June 1988, ELSEVIER Science Publishers, North-Holland, 1989, pp. 545–553

[4] Misiurewicz P.: Podstawy techniki mikroprocesorowej, WNT, Warszawa 1981

[5] Niederliński A.: Systemy komputerowe automatyki przemysłowej, t. 1: Sprzęt i oprogramowanie, WNT, Warszawa 1984

[6] Orłowski H.: Komputerowe układy automatyki, WNT, Warszawa, 1987

[7] Plaza R.A., Wróbel E.J.: Systemy czasu rzeczywistego, WNT, Warszawa, 1988

Wskazówki dla Autorów

Nadsyłane artykuły nie mogą być publikowane lub przeznaczone do opublikowania w innych czasopismach.

Materiał oprócz tekstu zasadniczego powinien zawierać:

- krótki życiorys zawodowy Autora i jego zdjęcie,
- wykaz literatury,
- tabele,
- materiał ilustracyjny (rysunki, zdjęcia czarno-białe, wydruki) dołączony do artykułu (nie wklejać materiału w tekst),
- podpisy pod ilustracje.

Na osobnej stronie prosimy podać: tytuł naukowy, nazwisko i imię, nazwę zakładu pracy, numer telefonu służbowego oraz informację, jaką drogą przekazać honorarium – kasa Wydawnictwa, poczta, konto.

W związku z naliczaniem podatku dochodowego prosimy także podać:

- oprócz nazwiska i imienia – także drugie imię,
- imiona rodziców,
- datę i miejce urodzenia,
- numer identyfikacyjny PESEL,
- miejsce zameldowania,
- adres Urzędu Skarbowego właściwego dla miejsca zamieszkania Autora.

Tekst artykułu prosimy dostarczyć w formie maszynopisu lub wydruku komputerowego, pisanego jednostronnie, z podwójnym odstępem (30 wierszy na stronie i 60 znaków w wierszu).

Rysunki powinny być czytelne i mogą być wykonane ołówkiem.

Są wśród nas

Komputery osobiste stworzyły bardzo pomyślne warunki dla aktywizacji zawodowej środowisk inwalidzkich. Żeby jednak wykorzystać tę szansę w Polsce konieczna jest pomoc środowiska informatycznego i całego społeczeństwa. Krokiem w tym kierunku było utworzenie w 1990 r. w środowisku związanym z Instytutem Podstaw Informatyki PAN – Fundacji Pomocy Matematykom i Informatykom Niesprawnym Ruchowo. Jej celem jest wspomaganie matematyków i informatyków z uszkodzeniami narządów ruchu, tworzenie warunków umożliwiających uprawianie działalności zawodowej, samorealizację i rozwój osobowości, a także wspomaganie socjalne i materialne podopiecznych. Zarząd Fundacji tworzą osoby oddające swój czas i pracę na rzecz środowiska całkowicie społecznie.

Fundacja obejmuje obecnie swoją działalnością blisko 100 osób w wieku od 18 do 65 lat. Są wśród nich studenci, absolwenci wydziałów matematyki i informatyki wyższych uczelni oraz Policealnego Studium Informatyki dla Niesprawnych Ruchowo w Konstancinie, obecni i byli pracownicy naukowi uczelni i Polskiej Akademii Nauk, programiści i operatorzy komputerów. Większość podopiecznych Fundacji to osoby bardzo ciężko poszkodowane, na wózkach inwalidzkich, niektórzy z nich nie mogą w ogóle poruszać się bez pomocy osób trzecich. Sytuacja materialna podopiecznych jest w większości bardzo trudna, niekiedy wręcz tragiczna, dlatego też Fundacja przychodzi im z pomocą finansową i rzeczową.

Podstawowy cel – tworzenie warunków do pracy zawodowej – Fundacja realizuje przez organizowanie kursów o zróżnicowanej tematyce i poziomie, wypożyczanie sprzętu oraz przez szukanie zleceń na prace informatyczne. Realizacji tego celu służy Laboratorium Komputerowe. Sprzęt i oprogramowanie pochodzi z darów różnych organizacji i firm komputerowych. Pomocy udzieliło też wiele osób prywatnych. Dzięki KBN i Państwowej Fundacji Rehabilitacji Osób Niepełnosprawnych (PFRON) Fundacja jest w trakcie wykonywania przez podopiecznych materiałów dotyczących szkolenia informatycznego. Jest to jednocześnie szkolenie wykładowców i konsultantów (niepełnosprawnych) prowadzących pilotażowe zajęcia dydaktyczne i przygotowujących ich programy. Z Laboratorium stale korzysta ok. 70 osób, dorywczo ok. 200 rocznie.

Niestety, Fundacja nie zarabia, brak więc środków na pomoc socjalną dla podopiecznych, a także na pensje trzech osób niepełnosprawnych zatrudnionych w Laboratorium na dwóch etatach (księgowość, opieka nad Laboratorium, wypożyczanie sprzętu, programów, itp.). Jest też za mało – w stosunku do możliwości i potrzeb – zleceń informatycznych, chociaż wielu podopiecznych reprezentuje wysoki poziom umiejętności. Ze względu na inwalidztwo i obojętny na ogół stosunek firm i instytucji do osób niepełnosprawnych, nie są bowiem w stanie „sprzedać" tej wiedzy. Przyjmą chętnie zlecenia m.in. na prace programistyczne, przepisywanie tekstów, zakładanie i eksploatację baz danych, wykonywanie nalepek adresowych, skład komputerowy, nauczanie w domu. Część z podopiecznych ma już modemy, a więc może przesyłać prace pocztą elektroniczną. Laboratorium Komputerowe prowadzi na zlecenie kursy m.in. podstawowe w zakresie DOS i Windows, edytorów, baz danych, arkuszy kalkulacyjnych, programów F-K Saari i Simplex.

Każde zlecenie i każda pomoc finansowa jest ratunkiem dla podopiecznych Fundacji.

Zainteresowanym podajemy nr konta bankowego: **BGŻ I/O Warszawa 399003-10802047-2710.** Bliższe informacje na temat zleceń i kursów komputerowych można uzyskać pod warszawskimi telefonami: **40-00-71 w. 200** (Laboratorium Komputerowe, ul. Tatrzańska 7a, 00-742 Warszawa), **13-38-39 i 39-69-22** (członkowie Rady Fundacji).

Analiza bezpieczeństwa komputerowego systemu sterowania

Janusz Górski
Andrzej Wardziński
Francusko-Polska Wyższa Szkoła
Nowych Technik Informacyjno-Komunikacyjnych
Poznań

Termin bezpieczeństwo dotyczy wpływu systemu na środowisko zewnętrzne. System jest oceniany jako niebezpieczny, gdy jego funkcjonowanie może mieć negatywne skutki dla środowiska zewnętrznego. Jako skutki negatywne rozumiemy – na przykład – straty materialne (zwykle większe od wartości samego systemu), zniszczenie maszyn lub innych obiektów, spowodowanie eksplozji lub zranienie ludzi czy też nawet ich śmierć. Bezpieczeństwo jest definiowane jako *gwarancja, że zastosowanie systemu nie doprowadzi do katastrofy*. Jedną z dziedzin, w której zagadnienie bezpieczeństwa jest bardzo istotne, jest medycyna. Współcześnie wiele systemów medycznych jest sterowanych przez komputery, niektóre z nich nawet bez bezpośredniego nadzoru człowieka. Stąd wymóg, aby takie systemy zapewniały bezpieczeństwo, co w tym przypadku oznacza, że ich zastosowanie nie może powodować negatywnych skutków dla zdrowia pacjentów.

W niniejszym artykule pokazano przykład takiego systemu. Przedstawiono zagadnienie analizy bezpieczeństwa. Opisano dwie metody analizy bezpieczeństwa: FMEA (*Failure Mode and Effect Analysis*) oraz analizę drzew błędów – FTA (*Fault Tree Analysis*). Pokazano przykład zastosowań tych metod dla wybranego zagrożenia występującego w omawianym systemie. Wskazano na niebezpieczeństwa wynikające z braku jednoznaczności tych metod. Przedstawiono zalety formalizacji wyników analizy bezpieczeństwa oraz przykładową formalizację drzewa błędów.

System Therac-25

Przykładem systemu medycznego jest Therac-25 [6], służący do leczenia nowotworów przez zastosowanie promieniowania X lub wiązki elektronów. Zaletą leczenia za pomocą promieniowania jest ograniczenie obszaru działania tylko do chorej tkanki objętej polem wiązki. Wpływ promieniowania na zdrową tkankę, będącą poza wiązką jest minimalny. Stosuje się dwa rodzaje terapii zależnie od umieszczenia nowotworu w organiźmie. Wiązka elektronów jest bardzo skuteczna w niszczeniu nowotworów znajdujących się blisko powierzchni skóry, natomiast promieniowanie X jest stosowane dla nowotworów w głębi ciała pacjenta.

System Therac-25 został wyprodukowany przez AECL (*Atomic Energy of Canada Limited*). Główną działalnością AECL jest projektowanie i instalacja reaktorów nuklearnych.

Podstawową częścią systemu Therac-25 jest akcelerator liniowy generujący wiązkę elektronów (dla pierwszego typu terapii) lub fotonów. Dla drugiego rodzaju terapii wiązka fotonów jest przekształcana w wiązkę promieniowania X. Cały przebieg terapii jest nadzorowany przez komputer, który steruje wszystkimi urządzeniami zgodnie z poleceniami operatora. Podczas terapii operator ze względów bezpieczeństwa znajduje się poza pomieszczeniem, w którym pacjent jest poddawany napromieniowaniu. System zapewnia dwustronną łączność dźwiękową przez interkom, a przez cały czas operator może obserwować pacjenta na ekranie monitora video.

Prototyp akceleratora Therac-25 opracowano w 1976 r. W 1983 r. powstała wersja komercyjna, w której zastosowano komputer sterujący całym urządzeniem. Użyto komputera PDP-11, do którego był przyłączony terminal VT100, którym posługiwał się operator. Całe oprogramowanie komputera zostało napisane w asemblerze i nie przekracza 32 KB wykonywalnego kodu. Cechą szczególną Therac-25 jest to, że jego oprogramowanie jest całkowicie odpowiedzialne za poprawne funkcjonowanie systemu – nie ma dodatkowego sprzętowego systemu kontroli funkcjonowania systemu.

Podczas jednego zabiegu (zwykle leczenie obejmowało cykl kilku lub kilkunastu zabiegów) pacjent otrzymywał bezpieczną dla zdrowia dawkę poniżej 200 rad (zwykle kilkadziesiąt rad). Jako niebezpieczną jednorazową dawkę (przyjętą przez cały organizm uważa się 500 rad – szansa przeżycia takiej dawki wynosi ok. 50%.

Wyprodukowano 11 komercyjnych egzemplarzy systemu Therac-25, z których pięć zainstalowano w USA, a pozostałe sześć w Kanadzie. W latach 1985–1987 nastąpiło sześć wypadków nadmiernego napromieniowania. W każdym z tych przypadków pacjent otrzymał dawkę rzędu 10 000–20 000 rad, co powodowało śmierć w ciągu kilku tygodni lub miesięcy od wypadku. W lutym 1987 r. FDA[1] uznała system za niebezpieczny (niezgodny z prawem dopuszczającym do użytku sprzęt medyczny) i zakazała jego eksploatacji. Jednocześnie zobowiązano producenta do wykonania **analizy bezpieczeństwa** systemu Therac-25 oraz modyfikacji koniecznych dla zapewnienia bezpieczeństwa.

Analiza bezpieczeństwa

Analiza bezpieczeństwa polega na tworzeniu modelu systemu (lub jego części) oraz wykazaniu (na podstawie utworzonego modelu) cech systemu istotnych dla jego bezpieczeństwa.

Analiza jest wykonywana przez eksperta znającego dokładnie zasady działania badanego systemu. Metoda ta może być stosowana zarówno na etapie projektowania systemu, jak i do niezależnej od wytwórcy oceny istniejącego systemu.

Analiza to część szerszego procesu – **oceny bezpieczeństwa.** Ocena zmierza do określenia stopnia bezpieczeństwa systemu. Podstawą procesu oceny są informacje uzyskane z:

- pomiarów funkcjonującego systemu,

[1] FDA Food and Drug Administration – ministerstwo Stanów Zjednoczonych odpowiedzialne m.in. za dopuszczanie do użytku sprzętu medycznego.

Prof. dr hab. inż. JANUSZ GÓRSKI ukończył studia w 1970 r. na Wydziale Elektroniki Politechniki Gdańskiej. W 1976 r. uzyskał stopień doktora, a w 1990 doktora habilitowanego w dziedzinie informatyki. Od 1992 r. pracuje we Francusko-Polskiej Wyższej Szkole Nowych Technik Informatyczno-Komunikacyjnych w Poznaniu, gdzie jest profesorem oraz kierownikiem Zakładu Informatyki i Zespołu Badawczego Inżynierii Oprogramowania. Jest autorem licznych publikacji. Kierował pracami nad wytwarzaniem oprogramowania do zastosowań w dziedzinach sterowania procesami przemysłowymi, telekomunikacji i baz danych. Jego zainteresowania badawcze skupiają się na inżynierii oprogramowania oraz bezpieczeństwie, poufności i niezawodności systemów komputerowych. Jest przewodniczącym komitetu EWICTS TC7 oraz członkiem zespołu WG5.4 IFIP.

- odniesienia do doświadczeń z poprzednich systemów,
- analizy bezpieczeństwa.

W przypadku systemu Therac-25, w analizie bezpieczeństwa wykorzystano trzy metody:

- niezależną analizę kodu, wykonaną przez niezależnego eksperta,
- *Failure Mode and Effect Analysis* (FMEA),
- *Fault Tree Analysis* (FTA).

Analiza bezpieczeństwa systemu Therac-25 zajęła prawie dwa lata. W tym czasie producent systemu sześć razy przesyłał wyniki do FDA, gdzie je analizowano i za każdym razem wskazywano niejednoznaczne i niejasne fragmenty analizy oraz proponowano wprowadzenie zmian poprawiających jakość analizy. Długość procesu analizy bezpieczeństwa oraz liczba nawrotów świadczą o tym, jak trudne jest osiągnięcie jednoznaczności i precyzji, niezbędnych dla zapewnienia wiarygodności oceny bezpieczeństwa systemu.

Wynikiem końcowej analizy bezpieczeństwa była propozycja 10 zmian konstrukcyjnych systemu, znaczących zmian w oprogramowaniu oraz udoskonalenie dokumentacji użytkowej i technicznej. Czas wprowadzenia zmian sprzętowych oszacowano na pół roku, a zmian programowych – na jeden rok. Podkreśla się, że wysoka jakość analizy bezpieczeństwa była w znacznym stopniu wynikiem niezależnej oceny tego procesu przez FDA, co pozwoliło na wyeliminowanie niejednoznaczności i braku precyzji.

Przed omówieniem metod FMEA i FTA konieczne jest przedstawienie jednego istotnego elementu konstrukcji systemu Therac-25 – jest nim obrotowy stół, w którym są zamontowane urządzenia wykorzystywane w różnych typach zabiegów. Stół ten znajduje się przez cały czas trwania zabiegu nad pacjentem. Stół zawiera dwa istotne urządzenia wykorzystywane podczas zabiegów. Aby mogły się one znaleźć w odpowiedniej pozycji (nad pacjentem) stół jest obracany przez silnik elektryczny sterowany komputerem. Jednym z wbudowanych w stół urządzeń jest zespół elektromagnesów, wykorzystywany podczas terapii wiązką elektronów jako „soczewka" dla wiązki. Drugie urządzenie służy do przekształcenia wiązki fotonów w jednorodną wiązkę promieniowania X. Proces ten wymaga dużych energii a otrzymywana wiązka promieniowania ma około 100-krotnie mniejszą energię niż padająca wiązka fotonów. Oznacza to, że jeżeli podczas zabiegu z wykorzystaniem promieniowania X stół będzie się znajdował w nieprawidłowej pozycji, pacjent otrzyma ok. 100-krotnie większą dawkę promieniowania od dopuszczalnej. Jest to jeden ze scenariuszy wypadków nadmiernego napromieniowania.

Metoda FMEA
Failure Mode Effect Analysis

Metoda FMEA jest stosowana do zbierania i systematyzacji wiedzy o komponentach systemu [1]. Jako metoda *bottom-up* wymaga od osoby wykonującej analizę szczegółowej wiedzy, dotyczącej zasad działania systemu i jego elementów. Wszystkie uzyskane informacje są przedstawiane za pomocą tabelki o kolumnach typowych dla tej metody. Dla każdego komponentu systemu jest powtarzana następująca procedura. W pierwszej kolumnie jest wpisywana nazwa (identyfikator) *elementu*, a w drugiej jego *funkcja*. W kolejnym kroku ustalane są wszystkie możliwe awarie lub możliwości nieprawidłowego działania danego elementu (kolumna *typ awarii*). Dla każdej awarii jest określany mechanizm dojścia do awarii (przyczyna), przy czym może istnieć wiele sposobów spowodowania jednego typu awarii. W kolumnie *efekt* opisywane są skutki awarii dla innych elementów. Następnie są identyfikowane *metody wykrywania awarii*. Ostatnia kolumna zawiera *uwagi* dotyczące awarii.

Tabela FMEA pozwala na określenie zależności między funkcjonowaniem poszczególnych elementów systemu oraz identyfikację komponentów, których prawidłowe działanie jest krytyczne dla całego systemu.

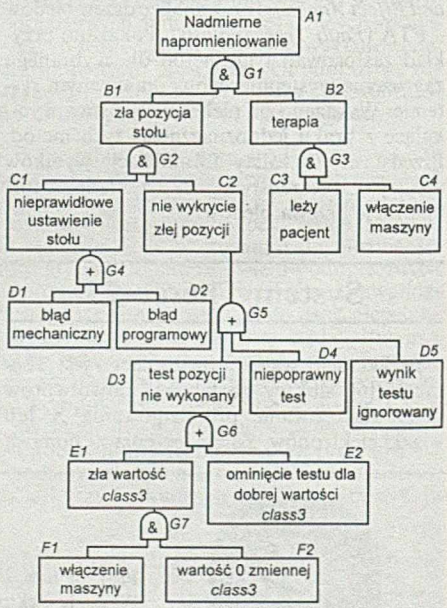
Tabela zawiera uproszczony przykład zastosowania metody FMEA dla dwóch elementów systemu Therac-25, silnika obracającego stołem oraz mikroprzełącznika służącego do wykrywania pozycji stołu.

Przykład tabeli FMEA dla systemu Therac-25

Element	Funkcja	Typ awarii	Przyczyna	Efekt	Metoda wykrycia	Uwagi
Silnik	obraca stołem	nie działa	przepalił się	stół nie obraca się	pośrednie: przez detektory pozycji stołu, *time-out*	system nie może pełnić swojej misji
			brak napięcia			
Mikroprzełącznik	detektor pozycji stołu	brak styku	pęknięcie sprężyny	brak sygnalizacji pozycji stołu	*time-out*	system nie może pełnić swojej misji
		zwarcie	przepalenie się	nieprawidłowa sygnalizacja pozycji	niespójność sygnalizacji	

Metoda FTA
Fault Tree Analysis

FTA (analiza drzew błędów) jest metodą typu *top-down* – analiza jest rozpoczynana od identyfikacji hazardów, których wystąpienie jest możliwe w danym systemie. Głównym celem metody FTA jest identyfikacja zbioru zdarzeń, które mogą spowodować wystąpienie *hazardu* [4]. Jako *hazard* rozumiany jest stan systemu prowadzący do katastrofy. Dla każdego hazardu tworzone jest oddzielne *drzewo błędów*. Na rys. 1 zostało przedstawione przykładowe drzewo błędów dla sytuacji, gdy nastąpiło nadmierne napromieniowanie pacjenta, spowodowane złą pozycją stołu. Należy zwrócić uwagę na fakt, że w drzewie są przedstawione zarówno zdarzenia sprzętowe i programowe, jak i zdarzenia zewnętrzne, takie jak działanie operatora. Możliwość analizy całego systemu jest jedną z podstawowych zalet analizy drzew błędów.



Rys. 1. Przykład drzewa błędów systemu Therac-25

Podstawowym elementem drzewa błędów jest zdarzenie (ang. *event*). Hazard, dla którego tworzone jest drzewo błędów, jest określany często jako zdarzenie nadrzędne (ang. *top-event*). Dla hazardu określane są wszystkie znane przyczyny bezpośrednie. Przyczyny są

Mgr inż. ANDRZEJ WARDZIŃSKI dyplom magistra inżyniera informatyki uzyskał w 1993 r. na Wydziale Elektroniki Politechniki Gdańskiej. Od 1993 r. jest zatrudniony jako pracownik naukowo-dydaktyczny we Francusko-Polskiej Wyższej Szkole Nowych Technik Informatyczno-Komunikacyjnych w Poznaniu. Zainteresowania naukowe obejmują zastosowanie metod formalnych do specyfikacji i weryfikacji oprogramowania oraz metody analizy bezpieczeństwa i ich wzbogacenie o formalną semantykę.

łączone z hazardem (na rysunku) za pomocą *bramy AND* lub *OR*. Brama *AND* oznacza, że muszą równocześnie wystąpić wszystkie przyczyny powodujące zdarzenie wyjściowe, natomiast brama OR oznacza, że dla zaistnienia zdarzenia wyjściowego wystarczy jedna z przyczyn wejściowych. Kolejne kroki analizy polegają na poszukiwaniu przyczyn znanych zdarzeń. Cały proces jest kontynuowany aż do osiągnięcia żądanego poziomu szczegółowości lub stwierdzenia braku możliwości kontynuacji, gdyż dane zdarzenie jest zdarzeniem zewnętrznym i nie można określić jego przyczyn.

W przykładowym drzewie (rys. 1) hazardem jest zdarzenie *nadmierne napromieniowanie*. Jest ono powodowane przez jednoczesne wystąpienie *złej pozycji stołu* i *terapii*. Terapia następuje wtedy, gdy *pacjent leży* na swoim miejscu i następuje włączenie maszyny. Zdarzenie *zła pozycja stołu* jest spowodowane przez *nieprawidłowe ustawienie stołu* oraz *niewykrycie złej pozycji* do chwili rozpoczęcia terapii. *Nieprawidłowe ustawienie stołu* może być skutkiem *błędu mechanicznego* lub *programowego*. Przed rozpoczęciem terapii, oprogramowanie powinno sprawdzić pozycję stołu, więc *niewykrycie złej pozycji* może nastąpić tylko wtedy, gdy *test nie będzie wykonany*, gdy zostanie on *niepoprawnie wykonany* lub gdy *jego wynik będzie zignorowany*. W oprogramowaniu systemu Therac-25 zdefiniowana jest zmienna *class3*; jej wartość różna od zera wskazuje, że powinien być wykonany test pozycji stołu. *Zła wartość class3* oznacza, że w chwili *włączenia maszyny* zmienna *class3 ma wartość 0*.

W oprogramowaniu systemu Therac-25 znajduje się procedura *Set-Up Test*, która sprawdza, czy został wciśnięty klawisz włączający maszynę, a następnie – w zależności od wartości zmiennej *class3* – wywołuje funkcję testu pozycji stołu (gdy *class3* $\neq 0$). Zmienna *class3* jest inicjowana w programie do wartości 0. *Set-Up Test* zwiększa wartość *class3* przy każdym wywołaniu procedury. Problem polega na tym, że procedura *Set-Up Test* jest wielokrotnie wywoływana i za każdym razem wartość *class3* jest zwiększana o 1. Doprowadza to w końcu do przepełnienia. Dodanie jedynek do wartości 255 (zmienna *class3* zajmuje 1 bajt, czyli 8 bitów) powoduje, że zmienna *class3* przyjmuje wartość 0. Jeżeli w tym momencie operator wciśnie klawisz, to zabieg zostanie rozpoczęty bez sprawdzenia pozycji stołu. Błąd jest banalny (zwykła pomyłka programisty), jednak występuje w okolicznościach znacznie utrudniających jego wykrycie przez testowanie. Przykład ten pokazuje skuteczność oraz siłę metod analitycznych w porównaniu z testowaniem.

Powyższy opis słowny jest lepiej zrozumiały niż rysunek drzewa błędów, ale niektóre szczegóły mogą w dalszym ciągu budzić wątpliwości. Dla precyzyjnego i jednoznacznego przedstawienia przykładowego drzewa błędów systemu Therac-25 są potrzebne jeszcze dodatkowe informacje. Jedną z nich jest fakt, że system ma zabezpieczenie uniemożliwiające stołowi zmianę pozycji w trakcie terapii. Gwarantuje to, że po ustawieniu stołu (poprawnym lub złym) przez cały czas terapii stół pozostanie w tej samej pozycji (jest to założenie analizy). Zabezpieczenie to jest krytyczne dla bezpiecznego przeprowadzenia leczenia. Należy też sprecyzować wiedzę dotyczącą terapii. Czas jednego zabiegu jest ograniczony i w zależności od danych określanych przez lekarza, wynosi kilka lub kilkanaście sekund. Zakłada się, że przez cały czas zabiegu pacjent będzie znajdował się w wyznaczonym miejscu i nie zmieniał pozycji ciała. Od rozpoczęcia terapii do pojawienia się napromieniowania mija ściśle określony czas, niezbędny systemowi na uruchomienie odpowiednich urządzeń, takich jak akcelerator. Oczywiste jest (ale warte odnotowania), że dla spowodowania hazardu terapia i zła pozycja stołu muszą wystąpić jednocześnie.

Istotne jest też rozróżnienie przyczyn wystarczających oraz koniecznych poszczególnych zdarzeń. Oczywistą cechą systemu jest to, że dwa zdarzenia: zła pozycja stołu i terapia są *wystarczającymi* przyczynami hazardu, ale *koniecznym* zdarzeniem jest tylko terapia (jeżeli nie będzie pacjenta – nie może zdarzyć się hazard). Natomiast zła pozycja stołu nie jest konieczna – możliwe są inne scenariusze wypadków (nie omawiane w tym przykładzie), np. złe sterowanie terapią spowodowane błędem oprogramowania.

Istotne dla analizy drzew błędów kolejne informacje dotyczą konstrukcji systemu lub procedury jego używania. Wiele hazardów pojawia się wtedy, gdy system działa niezgodnie z określoną dla niego procedurą. W przypadku systemu Therac-25, każdy zabieg odpowiada następującemu scenariuszowi: pacjent zajmuje miejsce na stole, ustawiany jest stół z przyrządami (w złej pozycji, jeżeli ma nastąpić wypadek), następnie operator wciska klawisz włączający maszynę, co powoduje wykonanie testu pozycji stołu (lub pominięcie testu spowodowane przez opisany błąd programowy) oraz rozpoczęcie terapii. Należy zwrócić uwagę na fakt, że drzewo błędów opisuje te fragmenty procedury, które są zagwarantowane przez konstrukcję systemu Therac-25. W przykładowym drzewie dwa razy występuje zdarzenie *włączenie maszyny*, gdyż procedura określa, że jest to przyczyna zarówno wykonania (lub pominięcia) testu pozycji, jak i rozpoczęcia terapii.

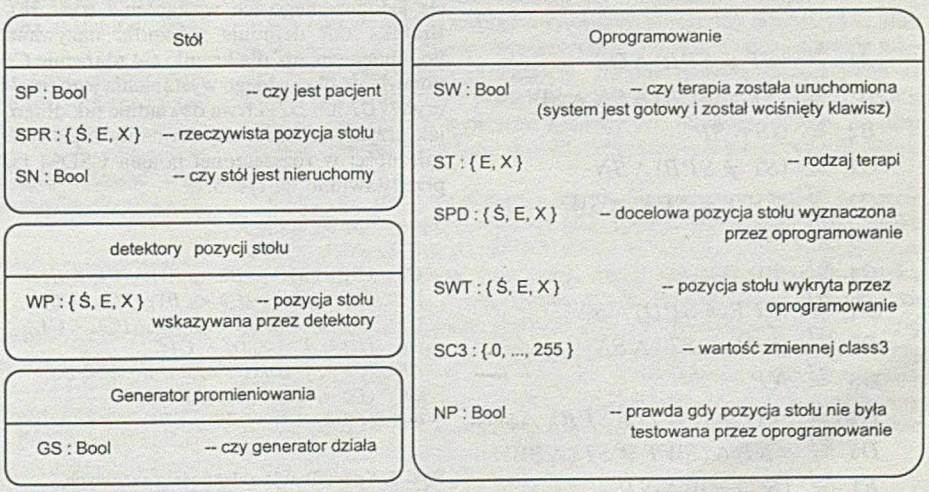
Formalizacja i jej cele

Powyższy przykład pokazuje, jak wiele informacji jest potrzebnych do precyzyjnego przedstawienia wyników analizy drzewa błędów. Świadczy to o tym, że sam rysunek nie niesie wystarczającej informacji do zrozumienia wyników analizy i dlatego niezbędny dla przedstawienia wyników analizy jest opis drzewa. Zwykle taki opis jest przedstawiany za pomocą języka naturalnego (tak, jak uczyniono to opisując drzewo błędów systemu Therac-25). Jedną z istotnych cech języka naturalnego jest jego niejednoznaczność. Powoduje to niebezpieczeństwo złej interpretacji wyników analizy. Zapewne również powyższy opis zawiera wiele niejednoznaczności. Podobną wadą jest obarczona metoda FMEA, której tabelka jest opisana wyłącznie za pomocą języka naturalnego. Sposobem usunięcia tej wady jest narzucenie jednoznacznej interpretacji opisu wyników analizy, czyli formalizacja jej semantyki.

Zastosowanie formalnej semantyki do opisu wyników analizy pozwoli osiągnąć:

- jednoznaczność wyników analizy,
- uniezależnienie się od eksperta podczas interpretacji wyników analizy,
- niezależną kontrolę procesu analizy,
- wprowadzenie standaryzacji metod analizy,
- zastosowanie narzędzi programowych wspierających proces analizy oraz kontrolujących poprawność wyników,
- definiowanie warunków do monitorowania stanu systemu,
- integrację wyników różnych metod analizy.

Zastosowanie formalnej semantyki może w znacznym stopniu **podnieść jakość procesu analizy**, co w efekcie da większą wiarygodność wyników. Istotną zaletą jest też możliwość automatyzacji przedstawiania wyników analizy oraz znacznego szerszego ich wykorzystania. Oczekuje się, że w przypadku integracji wielu metod, ich łączny wynik przewyższy sumę wyników osiąganych przez poszczególne metody.



Symbole oznaczające typ terapii lub odpowiadającą jej pozycję stołu:
E - tryb wiazki elektronów,
X - tryb promieniowania X,
Ś - pozycja stołu ze światłem nad pacjentem.

Rys. 2. Niektóre obiekty systemu Therac-25

Należy podkreślić, że celem nie jest formalizacja metod analizy bezpieczeństwa, a jedynie wyników osiąganych przez ich nieformalne zastosowanie. W przypadku metody FTA proponuje się, aby żaden formalizm nie ograniczał analityka podczas tworzenia drzewa błędów [4].

Przykład opisu formalnego drzewa błędów

W przedstawionym za pomocą rysunku drzewie błędów istotne informacje pozostają opisane w niejednoznaczny sposób. Bramki **AND** i **OR** nie przedstawiają wszystkich zależności między zdarzeniami. Również same nazwy zdarzeń użyte w drzewie błędów mogą być uznane za niejednoznaczne.

Prace nad opracowaniem formalnej semantyki, służącej do opisu wyników drzew błędów i innych metod analizy bezpieczeństwa, zainicjowano stosunkowo niedawno [2].

Specyfikacja zdarzeń

Jednoznaczny opis zdarzeń występujących w drzewie błędów wymaga odwołania się do niektórych elementów konstrukcji systemu. W naszym przykładzie zdarzenia można opisać za pomocą atrybutów obiektów, takich jak: stół, oprogramowanie lub detektory pozycji stołu. Stan każdego z tych obiektów w danej chwili jest określony przez wartości jego *atrybutów*, np. stan *generatora promieniowania* jest określony przez wartość atrybutu *działanie*, który przyjmuje dwie wartości: *prawda*, gdy generator jest włączony (wytwarza wiązkę), lub *fałsz*, gdy jest on wyłączony. Konstrukcja systemu opisanego przez obiekty i ich atrybuty przedstawiono na rys. 2.

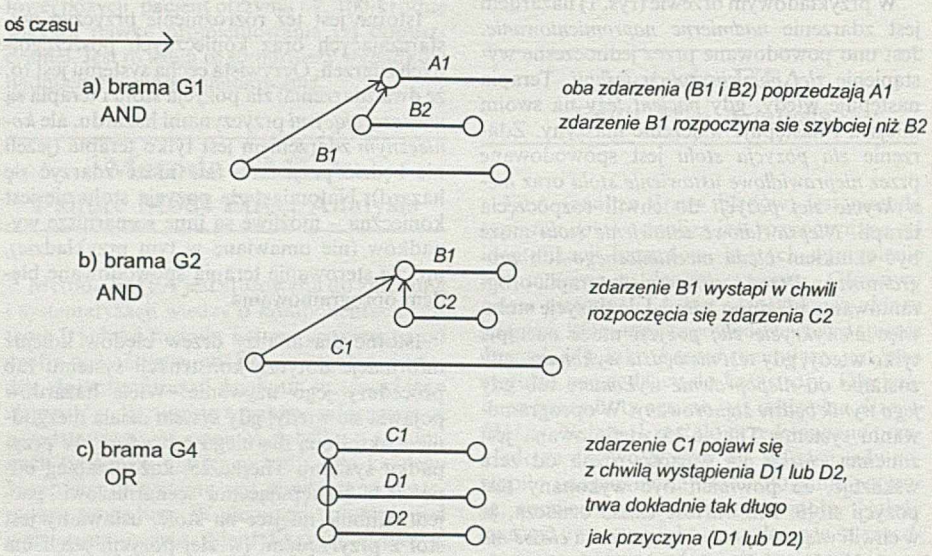
Aby opisać zdarzenie jest definiowane wyrażenie logiczne, którego zmiennymi są atrybuty zdefiniowane w systemie. Przyjmuje się, że zdarzenie zachodzi w chwili t, jeżeli wartość wyrażenia w chwili t wynosi *prawda*. Na rys. 3 przedstawiono definicje zdarzeń występujących w drzewie błędów systemu Therac-25.

$$
\begin{aligned}
A1 &\stackrel{def}{=} (ST \neq SPR) \wedge GS \\
B1 &\stackrel{def}{=} (ST \neq SPR) \wedge SN \wedge SW \\
B2 &\stackrel{def}{=} GS \wedge SP \\
C1 &\stackrel{def}{=} (ST \neq SPR) \wedge SN \\
C2 &\stackrel{def}{=} (SWT \neq SPR) \wedge SW \\
C3 &\stackrel{def}{=} SP \\
C4 &\stackrel{def}{=} SW \\
D1 &\stackrel{def}{=} (SPR \neq SPD) \wedge SN \\
D2 &\stackrel{def}{=} (SPD \neq ST) \wedge SN \\
D3 &\stackrel{def}{=} NP \\
D4 &\stackrel{def}{=} \neg NP \wedge (SWT \neq SPR) \\
D5 &\stackrel{def}{=} \neg NP \wedge (SWT \neq ST) \wedge SW \\
E1 &\stackrel{def}{=} (SC3 = 0) \wedge NP \\
E2 &\stackrel{def}{=} (SC3 \neq 0) \wedge NP \\
F1 &\stackrel{def}{=} SW \\
F2 &\stackrel{def}{=} SC3 = 0
\end{aligned}
$$

Rys. 3. Specyfikacja zdarzeń

Specyfikacja zależności czasowych

Drugim elementem opisu drzewa błędów, wobec którego wymagana jest jednoznaczność, są zależności czasowe między zdarzeniami. Zależności te wynikają z własności bramek *AND* i *OR* przedstawionych na rys. 1. Ponadto, znajomość zasad funkcjonowania systemu Therac-25 pozwala precyzyjnie określić, które zdarzenia mogą wystąpić jednocześnie, które kolejno, w jakiej kolejności itp.



Rys. 4. Graficzna specyfikacja zależności czasowych

Typowe zależności czasowe zdarzeń przykładowego drzewa błędów przedstawiono na rys. 4. Każda pozioma kreska oznacza wystąpienie danego zdarzenia, gdzie lewy koniec to początek, a prawy – koniec danego zdarzenia. Strzałki określają przyczynowość (jedno zdarzenie powoduje wystąpienie innego zdarzenia). W przykładowym drzewie błędów można wyróżnić dwa rodzaje bramek *AND*: gdy skutek pojawia się jakiś czas po wystąpieniu obu przyczyn (rys. 3a) lub w momencie jednoczesnego ich wystąpienia (rys. 3b). Bramka *OR* definiuje zależność nazywaną uogólnieniem, np. dla bramki *G4* zdarzenie *C1* pojawia się dla każdego wystąpienia jego przyczyny (*D1* lub *D2*) i trwa dokładnie tak długo, jak przyczyna. Skrócony formalny zapis tych zależności w rozszerzonej notacji CSDM [5] przedstawiono na rys. 5.

$$
\begin{aligned}
G1&: (B1 \prec_H A1) \wedge (B2 \prec_H A1) \wedge (B1_s < B2_s) \\
G2&: (C1 \prec_H B1) \wedge (C2 \prec_H B1) \wedge (C2_s = B1_s) \\
G3&: (C3 \prec_H B2) \wedge (C4 \prec_H B2) \wedge (C3_s < C4_s) \\
G4&: (D1 =_e C1) \vee (D2 =_e C1) \\
G5&: (C2 =_e D3) \vee (C2 =_e D4) \vee (C2 =_e D5) \\
G6&: (D3 =_e E1) \vee (D3 =_e E2) \\
G7&: (F2 \prec_H E1) \wedge (SW \prec_H E1)
\end{aligned}
$$

Rys. 5. Specyfikacja zależności czasowych

Rysunki 3 i 5 tworzą razem formalny opis przedstawionego na rys. 1 drzewa błędów (dla systemu zdefiniowanego na rys. 2).

* * *

Jedną z dróg prowadzących do zwiększenia bezpieczeństwa zastosowań systemów komputerowych jest polepszenie jakości metod i technik tradycyjnie używanych podczas analiz bezpieczeństwa systemu. Podstawowa idea takiego podejścia polega na tym, aby zamiast podejmowania prób odkrycia i zaproponowania narzędzi całkowicie nowych, ulepszyć narzędzia już stosowane. Dzięki temu inżynier-analityk nie musi w sposób rewolucyjny zmieniać własnego zestawu narzędziowego, do którego jest on przyzwyczajony.

Formalizacja semantyki jest jednym ze sposobów uzyskania takiego właśnie efektu, na co wskazano w niniejszym artykule. Prace w tym zakresie zostały podjęte również przez innych autorów [3]. Zaprezentowane tu badania są obecnie kontynuowane w powiązaniu z projektem badawczym SHIP (*Safety of Hazardous Industrial Processes*) będącym częścią programu *Environment*, finansowanego przez Komisję Wspólnot Europejskich.

LITERATURA

[1] Analysis Techniques for System Reliability – Procedure for Failure Mode and Effect Analysis. International Electrotechnical Commission, IEC Standard, Publication 812, 1990

[2] Bloomfield R.E., Cheng J.H., Górski J.: Safety Analysis: a feasibility study into the development of a generic Safety Description Method (SDM). EUREKA Projekt SEW263 Adelard, 1990

[3] Bruns G., Anderson S.: Validating Safety Models with Fault Trees. SAFECOMP'93, Springer-Verlag, 1993

[4] Fault Tree Handbook. United States Nuclear Regulatory Commission, NUREG-0492, 1981

[5] Górski J.: Extending Safety Analysis Techniques with Formal Semantics. SSS'94, Birmingham, UK, 1994

[6] Leveson N.G., Turner C.S.: An Investigation of the Therac-25 Accident. IEEE Computer, July 1993.

Projektowanie oprogramowania systemów czasu rzeczywistego
Podstawowe fazy projektu

Zbigniew Huzar
Centrum Informatyczne
Politechnika Wrocławska

Systemy czasu rzeczywistego określa się zwykle pośrednio przez wyliczenie pewnych charakterystycznych cech [3, 5, 9]. Punktem wyjścia jest stwierdzenie, że jest to system komputerowy sprzężony z określonym obiektem (systemem) zewnętrznym – swoim otoczeniem. System wymienia informację z otoczeniem za pośrednictwem wspólnego interfejsu. Istotą prawidłowej współpracy jest wymiana właściwych danych zgodnie z odpowiednimi ograniczeniami czasowymi – system jest odpowiedzialny za nadążanie własnymi działaniami za działaniami swojego otoczenia. Konsekwencje nienadążenia – w przypadku systemów *słabo uwarunkowanych czasowo* – mogą prowadzić do błędów, których koszty wzrastają ze wzrostem czasu nienadążenia, natomiast w przypadku systemów silnie uwarunkowanych czasowo mogą nastąpić błędy, czasem nieprzewidywalne, a czasem nawet katastroficzne dla procesów lub ludzi.

Zasadnicze cechy, jakimi powinien charakteryzować się system czasu rzeczywistego omówione są poniżej.

Pierwszą – już wymienioną – cechą jest *odpowiednia szybkość działania*, dokładniej – spełnianie narzuconych wymagań czasowych. Należy przy tym podkreślić, że system jest zwykle sprzężony ze swoim otoczeniem wieloma kanałami (punktami interakcji, portami, bramkami), które mogą prowadzić jednoczesną, niezależną od siebie wymianę informacji, a system musi nadążać z jednoczesną obsługą tych kanałów. Stąd *współbieżność* działania jest nieodłączną cechą systemów czasu rzeczywistego.

Działanie systemu powinno być w pełni *przewidywalne* – chodzi o to, aby w każdych okolicznościach, pomimo niederministycznego zachowania się otoczenia systemu, można przewidzieć sposób działania systemu, w szczególności by można przewidzieć czasy wykonania odpowiednich czynności. Oczekuje się, że system będzie działać w zaplanowany z góry sposób, nawet w najgorszych, ale jawnie określonych, okolicznościach.

Prof. dr hab. inż. ZBIGNIEW HUZAR ukończył w 1969 r. studia na Wydziale Elektroniki Politechniki Wrocławskiej, pracę doktorską obronił w 1974, a habilitację uzyskał w 1990 r. za pracę „Programowanie procesów komunikujących się w czasie rzeczywistym". Obszarem jego zainteresowań jest programowanie współbieżne i systemy czasu rzeczywistego. Obecnie zajmuje stanowisko profesora nadzwyczajnego w Politechnice Wrocławskiej. Od ponad 10 lat jest dyrektorem Centrum Informatycznego Politechniki Wrocławskiej.

Wreszcie, system powinien być *niezawodny*, czyli działać właściwie, pomimo błędów i zakłóceń. Wymóg ten oczywiście ma swoje ograniczenia, gdyż nie można oczekiwać bezwzględnej niezawodności systemu, który z definicji, powinien składać się z zawodnych komponentów. W każdym przypadku stopień niezawodności systemu powinien być znany oraz powinna być określona zależność tej niezawodności od zawodności jego komponentów. W przypadku systemów, które tylko ograniczają, ale nie zaprzestają działania w przypadku niesprawności swych elementów składowych, powinien być określony wpływ takich niesprawności na obniżenie wydajności systemu.

Pomimo dużego znaczenia oraz wielu zastosowań, systemy czasu rzeczywistego nie doczekały się opracowania jednolitych, powszechnie akceptowanych modeli, a w ślad za tym systematycznych i uniwersalnych metod projektowania. W okresie ostatnich dziesięciu lat podjęto wiele prac, których celem jest wypracowanie inżynierii systemów czasu rzeczywistego. Szczególną rolę odgrywa tu inżynieria oprogramowania czasu rzeczywistego.

Model systemu czasu rzeczywistego

Pojęcie *systemu komputerowego* odnosimy ogólnie do systemu przetwarzającego informację. W takim określeniu – w sposób niejawny – zakłada się, że oprócz systemu istnieje jego otoczenie, z którym system prowadzi wymianę informacji. *Otoczenie* systemu oznacza zewnętrzne źródła oraz ujścia przetwarzanej informacji.

Najprostszym rodzajem są systemy *transformacyjne*, których schemat działania charakteryzuje to, że dla określonego zestawu danych wejściowych produkują określony wynik. Sprzężenie systemu z otoczeniem jest tu bardzo proste – charakteryzuje je jedno wejście i jedno wyjście. Związek pomiędzy danymi wejściowymi i danymi wyjściowymi można tu opisać przez określoną funkcję, czy – ogólniej – relację.

Bardziej złożonym rodzajem systemów są systemy *interaktywne* albo *reaktywne*. Ich działanie charakteryzuje wzajemne oddziaływanie między systemem a jego otoczeniem. Zwykle systemy interaktywne są systemami działającymi przez z góry nieograniczony czas. Opis związku między wejściami a wyjściami systemu jest bardziej złożony. System cechuje bowiem określony sposób zachowania, a nie tworzenie określonego wyniku.

Podstawą do sformułowania tego rodzaju związku będzie przyjęcie następującego modelu systemu [6]. System traktujemy jako trójkę składającą się ze zbioru *stanów S, zbioru interakcji A* oraz *zbioru obliczeń* Σ. Stan jest zestawem informacji, który jest pełną charakterystyką systemu odniesioną do określonej chwili. Interakcja jest wykonaniem niepodzielnej (atomowej) operacji, którą może wykonać system. Tutaj przyjmiemy ustalenie, że jedynymi interesującymi nas interakcjami będą operacje komunikacji, czyli wymiany *komunikatów* między systemem a jego otoczeniem. Przyjmujemy dodatkowo, że realizacja wymiany komunikatów wymaga synchronizacji systemu z otoczeniem. *Obliczenie* σ jest ciągiem, skończonym lub nieskończonym, postaci:

$$S_0 \xrightarrow{\alpha_1} S_1 \xrightarrow{\alpha_2} S_2 \qquad (1)$$

gdzie $s_i \in S$ $(i = 0, 1, 2...)$, $\alpha_i \in A$ $(i = 1, 2...)$. Jeżeli ciąg jest skończony, to kończy się pewnym stanem s_n. Obliczenie σ reprezentuje zmiany w systemie, w którym on poczynając od stanu początkowego s_0, *wykonuje kolejne interakcje (komunikacje między systemem a jego*

otoczeniem) $\alpha_1, \alpha_2 \ldots$, po których przechodzi odpowiednio do stanów s_1, $s_2 \ldots$ Zbiór Σ reprezentuje zbiór możliwych obliczeń systemu. Zatem związek pomiędzy wejściami a wyjściami systemu reaktywnego w pełni opisuje zbiór obliczeń Σ. Należy przy tym podkreślić, że traktowanie obliczenia tylko jako ciągu interakcji (podejście takie określa się mianem *przeplotu*) stanowi pewne uproszczenie, gdyż z punktu widzenia zewnętrznego obserwatora systemu mogą zajść sytuacje, gdy zachodzą jednocześnie interakcje na wielu bramkach.

System czasu rzeczywistego jest szczególnym rodzajem systemu interaktywnego – systemem, który charakteryzują momenty czasu, do których odnosimy stan systemu, oraz momenty czasu, w których zachodzą interakcje. Wprowadza się pojęcie zbioru *chwil czasowych TIME*. Zamiast stanu wprowadza się pojęcia *stanu czasowego* jako pary $\langle s, t \rangle$, gdzie $s \in S$ jest stanem systemu, zaś $t \in TIME$ jest chwilą, w której stan ten został osiągnięty. Zamiast o interakcji mówi się o *interakcji czasowej* rozumianej jako para $\langle \alpha, t \rangle$, gdzie $\alpha \in A$ jest interakcją, zaś $t \in TIME$ jest momentem wystąpienia tej akcji. Zbiór obliczeń czasowych Σ_T jest zbiorem ciągów postaci:

$$\langle S_0, t_0 \rangle \xrightarrow{\langle \alpha_1, t_1 \rangle} \langle S_1, t_1 \rangle \xrightarrow{\langle \alpha_2, t_2 \rangle} \langle S_2, t_2 \rangle \ldots \quad (2)$$

Dla uproszczenia będziemy dalej przyjmować, że interakcje są wykonywane natychmiast oraz, że zmiana stanu systemu następuje w momencie wystąpienia interakcji. Cały system czasu rzeczywistego jest opisany w pełni przez podanie zbioru obliczeń czasowych Σ_T. Uproszczenie, które polega na tym, że zachodzi w danym momencie czasu tylko jedna interakcja, w przypadku systemów czasu rzeczywistego nie zawsze może być akceptowane. Dlatego bardziej adekwatne jest przyjęcie, że obliczenie czasowe ma postać:

$$\langle S_0, t_0 \rangle \xrightarrow{\langle \Omega_1, t_1 \rangle} \langle S_1, t_1 \rangle \xrightarrow{\langle \Omega_2, t_2 \rangle} \langle S_2, t_2 \rangle \quad (3)$$

gdzie $\Omega_1, \Omega_2, \ldots \subseteq A$. Takie podejście odzwierciedla prawdziwą współbieżność w działaniu systemu.

Przedstawiony abstrakcyjny model systemu czasu rzeczywistego może być punktem wyjścia do przedstawienia metod specyfikowania oraz implementacji. Pomijamy tu omówienie ważnej własności strukturalizacji modeli, odsyłając w tej sprawie m.in. do [1, 8]. Pomijamy również omówienia różnych modeli czasu, odsyłając w sprawach ogólnych do [2], oraz w szczegółowych – do [7]. W dalszych rozważaniach zakładamy najprostszy model globalnego czasu uniwersalnego.

Wyróżnia się dwa podejścia do opisu zbioru obliczeń. Można je nazwać podejściem *konstruktywnym* lub *generacyjnym*, oraz *aksjomatycznym*, chociaż nazwy te – jak zobaczymy niżej – mogą być mylące. W podejściu aksjomatycznym, zbiór obliczeń opisuje się przez podanie zbioru aksjomatów ustalonego systemu logicznego. Zbiór tych obliczeń, które spełniają zadane aksjomaty wyznacza zbiór Σ_T. Aksjomaty mogą być formułowane w wybranym języku logiki, np. logiki temporalnej. Natomiast w podejściu konstruktywnym zbiór obliczeń jest generowany przez określony mechanizm, np. program pisany w odpowiednim języku, albo przez maszynę – automat, generujący wszystkie obliczenia. Bardzo często mechanizm taki bywa opisywany również w sposób aksjomatyczny.

Metoda aksjomatyczna prowadzi bezpośrednio do określonego mechanizmu wnioskowania o własnościach zbioru obliczeń. Jeżeli Ax jest zbiorem aksjomatów opisującym zbiór obliczeń Σ_T, zaś C jest własnością wyrażoną w języku tej samej logiki, to system spełnia własność C, wtedy i tylko wtedy, gdy potrafimy w danym systemie formalnym wyprowadzić C z Ax. Opis konstruktywny jest często bardziej odpowiedni niż opis aksjomatyczny, ponieważ języki programowania są właśnie specjalnie przeznaczone do opisu obliczeń, podczas gdy systemy formalne są wybierane ze względu na ich własności logiczne. Zwróćmy uwagę, że każdy opis konstruktywny jest wyrażany w języku, który ma w pewien sposób zdefiniowaną semantykę. Semantyka może być, m.in. opisana przez zbiór aksjomatów w pewnym systemie formalnym. W tym sensie opis konstruktywny może być uważany za pewnego rodzaju opis aksjomatyczny.

Cykl życia oprogramowania

Typowe podejście przy tworzeniu systemów jest oparte na schemacie:

specyfikacja → projekt → implementacja

Specyfikacja, opisując zachowanie systemu z punktu widzenia użytkownika, stanowi ekstensjonalną charakterystykę systemu. Specyfikacja ma stanowić jednoznaczny i wyczerpujący opis oczekiwanych własności systemu. Ma też określać, co należy zrobić, a nie, jak to należy zrobić. Stanowi pewnego rodzaju kontrakt między systemem a jego otoczeniem. Zakładając określone własności otoczenia, system powinien gwarantować odpowiednią wydajność – nadążność we współpracy z otoczeniem.

Projekt zawiera decyzje o sposobie przejścia od specyfikacji do implementacji za pomocą wybranych środków realizacyjnych. Projekt systemu powinien spełniać nie tylko wymogi specyfikacji, lecz dodatkowo zabezpieczać system przed skutkami zakłóceń w działaniu otoczenia, tj. w przypadku odstępstw otoczenia od zakładanego zachowania. Podstawową czynnością projektowania ogólnego jest określenie architektury oprogramowania, to znaczy:

- zdekomponowanie całego systemu na części składowe,
- określenie ich funkcji,
- określenie ich parametrów wydajnościowych,
- określenie wzajemnych sprzężeń.

Wreszcie *implementacja* polega na określeniu tego, jak działają poprzednio użyte komponenty. Specyfika implementacji systemów czasu rzeczywistego polega na tym, aby spełnić funkcjonalne i czasowe wymagania przy wykorzystaniu ograniczonej ilości zasobów o ograniczonych zdolnościach czasowych i wydajnościowych.

Wymienione trzy fazy składają się na cykl wytwarzania systemów. Fazy te następują w zasadzie jedna po drugiej, a także możliwe są nawroty. W praktyce każda z wymienionych faz może składać się z wielu etapów, przy czym niekiedy trudno jednoznacznie zaliczyć konkretny etap do jednej z faz. Przyjmuje się, że proces tworzenia systemu składa się z ciągu kroków:

$$S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow \ldots S_n,$$

z których pierwszy S_1 jest specyfikacją wymagań systemu, następne są kolejnymi jego rozwinięciami, a ostatni S_n stanowi końcową implementację. Schemat ten ukazuje względność rozumienia specyfikacji implementacji. Obiekt S_i może być traktowany jako implementacja S_{i-1}, i jako specyfikacja dla obiektu S_{i+1}.

Ze względu na relatywność znaczeń używanych terminów, wystarczy mówić o specyfikacji i o implementacji. Poniżej sprecyzujemy rozumienie pojęć specyfikacji i implementacji.

Specyfikacja systemu czasu rzeczywistego

Specyfikacja stanowi ekstensjonalną charakterystykę systemu i opisuje *zewnętrznie dostrzegalne efekty* jego działania. Możliwe są dwa sposoby rozumienia tych efektów [4].

Pierwszy zakłada, że obserwowalne są *zmiany stanów* (wartościowania globalnych zmiennych) programu. Takie podejście jest stosowane w specyfikacjach systemów transformacyjnych. W tym przypadku specyfikacja wyraża związek między stanem początkowym a stanem końcowym programu.

Drugi sposób zakłada, że obserwowalne są interakcje między programem a jego otoczeniem, zachodzące w trakcie wykonywania programu. W tym przypadku specyfikacja przedstawia zbiór ciągów interakcji, które może dostrzec zewnętrzny obserwator systemu. Takie rozumienie jest stosowne przy opisie programów reaktywnych, a więc i systemów czasu rzeczywistego.

Działanie systemów czasu rzeczywistego było opisane ciągiem postaci (2) lub (3). Z punktu widzenia zewnętrznego obserwatora systemu dostrzegane są tylko interakcje na bramkach łączących system z otoczeniem, natomiast stany wewnętrzne systemu nie są przez niego obserwowalne. Obserwator dostrzega więc ciągi postaci:

$$\langle \alpha_0, t_0 \rangle, \langle \alpha_1, t_1 \rangle, \ldots, \langle \alpha_2, t_2 \rangle \ldots \quad (4)$$

lub

$$\langle \Omega_0, t_0 \rangle, \langle \Omega_1, t_1 \rangle, \ldots, \langle \Omega_2, t_2 \rangle \ldots \quad (5)$$

Ciągi tej postaci nazywa się *zachowaniami czasowymi*. Zatem specyfikacja określa zbiór ciągów interakcji między systemem a jego otoczeniem, zachodzących w trakcie pracy systemu, czyli zbiór wszystkich zachowań czasowych *TBeh*. Zbiór *TBeh* powinien odzwierciedlać wszystkie oczekiwane własności systemu. W klasie systemów reaktywnych wyróżnia się dwie grupy własności: własności *bezpieczeństwa* (ang. *safety*) oraz własności *żywotności* (ang. *liveness*). Intuicyjnie, własność bezpieczeństwa oznacza, że nic złego nie wydarzy się podczas pracy systemu np. nie będzie przekroczone zadane opóźnienie w reakcji na ustalone interakcje), a własność żywotności – że w końcu wydarzy się coś dobrego (np. nie nastąpi trwałe zablokowanie obliczeń prowadzonych przez komponenty systemu, czyli nastąpi sytuacja, w której ewentualna

okresowa blokada zostanie usunięta). Są także inne rodzaje własności, których nie można zaliczyć do żadnej z dwóch wymienionych grup. Np. stwierdzenie, że prawdopodobieństwo tego, iż opóźnienie na daną interakcję nie przekroczy 0,1 ms jest mniejsze od 0,99, nie jest ani własnością bezpieczeństwa, ani żywotności.

W klasie systemów czasu rzeczywistego rozpatruje się własności bezpieczeństwa i *własności czasowe*, natomiast własności żywotności mają drugorzędne znaczenie. Wynika to stąd, że własności bezpieczeństwa odnoszą się do zachowań skończonych, podczas gdy własności żywotności odnoszą się do zachowań nieskończonych, a ponadto odpowiednio sformułowane własności czasowe mogą gwarantować własności żywotności. Przykładowo, mało przydatna jest własność *bezstronności* (ang. *fairness*), należąca do grupy własności żywotności.

Własności czasowe mogą odzwierciedlać trzy rodzaje relacji, jakie mogą zachodzić między interakcjami zachowań:

- *Relatywnego uporządkowania* – chodzi o zachowanie porządku działań zgodnie z upływem czasu. Zwykle ograniczenie się do tych relacji nie jest wystarczające do opisu systemów czasu rzeczywistego.
- *Względnych zależności czasowych* – chodzi o wyrażenie nie tylko kolejności, ale dodatkowo odstępu czasu między kolejnymi interakcjami. Np. pozwala to określić czas reakcji przy założeniu, że interakcje występujące w zachowaniach da się połączyć w pary typu akcja-reakcja.
- *Bezwzględnych zależności czasowych* – pod uwagę są brane zdarzenia wraz z chwilą, do której się one odnoszą. W tym przypadku czas jest parametrem działania systemu.

Inny sposób podziału własności czasowych, to *ograniczenia wydajnościowe i ograniczenia środowiskowe*. Pierwsze ograniczenia odnoszą się do szybkości reakcji systemu określone interakcje wejściowe, dotyczy to w szczególności szybkości reakcji w najbardziej krytycznych przypadkach. Drugie ograniczenia określają szybkość reakcji na interakcje, które wymusza otoczenie systemu.

Zgodnie z tym co powiedziano wcześniej na temat sposobów przedstawiania modelu systemu czasu rzeczywistego, możliwe jest konstruktywne (czyli wykonywalne) i aksjomatyczne (czyli logiczne) przedstawienie specyfikacji systemu. W zależności od postaci specyfikacji można stosować różne podejścia do:

- *walidacji* specyfikacji oraz
- *weryfikacji* specyfikacji, czyli stwierdzania zgodności specyfikacji z implementacją.

Problem walidacji odnosi się przede wszystkim do pierwszej ze specyfikacji S_1. Specyfikacja ta stanowi specyfikację wymagań, a o jej adekwatności decyduje rzeczywiście użytkownik. Chodzi tu o porównanie dwóch obiektów: obiektu sformalizowanego – specyfikacja S_1, z obiektem niesformalizowanym – dziedziną empiryczną. Teoretycznie chodzi o dwie kwestie. Pierwsza dotyczy *niesprzeczności* specyfikacji, natomiast druga – jej *zupełności*. Pojęcia niesprzeczności i zupełności są tu rozumiane w sensie semantycznym, tzn. odnoszą się do dziedziny interpretacji specyfikacji, którą ustala użytkownik.

Rozstrzygnięcia o niesprzeczności specyfikacji – w pewnym zakresie przynajmniej teoretycznie – można dokonać automatycznie. Natomiast rozstrzygnięcie zupełności można przeprowadzić tylko empirycznie.

W przypadku, gdy mamy do czynienia z obiektem formalnym i nieformalnym ich porównanie jest możliwe wyłącznie przez testowanie. Pojęcie testowania należy rozumieć tu w sposób następujący. Jeżeli specyfikacja jest wyrażona w sposób konstruktywny, oznacza to, że jest możliwe generowanie oczekiwanych zachowań czasowych systemu. Testowanie polega na przyjęciu określonej zasady generowania wybranych zachowań i na rozstrzyganiu o ich adekwatności na gruncie przyjętej dziedziny zastosowań. Jeżeli specyfikacja jest wyrażona w sposób logiczny, to zamiast zachowań należy wyprowadzić nowe własności, z własności stanowiących specyfikację, i rozstrzygać o ich adekwatności po ich zinterpretowaniu w dziedzinie zastosowania.

Implementacja

Pojęcie implementacji, rozumiane jako przejście między kolejnymi poziomami S_i a S_{i+1}, w przypadku systemów reaktywnych, a więc i systemów czasu rzeczywistego, jest dosyć subtelne [4, 6]. Oto niektóre związane z tym zagadnienia:

- Żaden system nie może funkcjonować poprawnie przy braku jakichkolwiek ograniczeń zachowania swego otoczenia. Powstaje pytanie, w jaki sposób w implementacji odzwierciedlać ograniczenia otoczenia bez wprowadzania dodatkowych, nielegalnych ograniczeń. Jeden ze sposobów rozwiązania tego problemu opiera się na pojęciu *odrzucenia*. System w danym stanie może nie być zdolny do uczestnictwa we wszystkich interakcjach czasowych. Interakcje takie nazywa się interakcjami odrzuconymi, a każde zachowanie, prowadzące do stanu, w którym są odrzucane określone interakcje, nazywa się zachowaniem odrzucanym.
- Pojęcie interakcji na poziomie specyfikacji i implementacji mogą być różne. Interakcja na poziomie specyfikacji może być implementowana przez ciąg interakcji na poziomie specyfikacji.
- Podobny problem może dotyczyć danych: ziarnistość ich przedstawiania na poziomie specyfikacji jest na ogół większa od ziarnistości na poziomie implementacji.

Możliwe są co najmniej trzy sposoby rozumienia implementacji względem danej specyfikacji. Niech $TBeh_i$ oznacza zbiór zachowań specyfikacji S_i.

1. Implementacja S_{i+1} może być **redukcją** specyfikacji S_i. Dotyczy to tych przypadków, w których implementacja S_{i+1} spełnia dwa następujące wymogi:
   - każde zachowanie S_{i+1} jest także dopuszczalnym zachowaniem S_i; czyli $TBeh_{i+1} \subseteq TBeh_i$,
   - każde zachowanie, które S_{i+1} odrzuca, jest również odrzucane przez S_i.
2. Implementacja S_{i+1} może być rozszerzeniem S_i. Związek ten zachodzi wówczas, gdy specyfikacja S_i jest specyfikacją częściową. W tym przypadku S_{i+1} dokonuje ustalenia tych zachowań, które są poza obszarem definicji S_i. Implementacja S_{i+1} spełnia dwa warunki:
   - każde działanie, które jest dopuszczalne przez S_i, jest także działaniem wykonywanym przez S_{i+1} (co nie wyklucza jednak wykonywania jeszcze dodatkowych działań), czyli $TBeh_i \subseteq TBeh_{i+1}$;
   - każde działanie, które S_{i+1} odrzuca, jest także odrzucane przez S_i (S_{i+1} nie może odrzucać więcej zachowań niż to czyni S_i).
3. Implementacja S_{i+1} może być **uściśleniem** S_i. W tym przypadku obie specyfikacje są równoważne ekstensjonalnie, tzn. reprezentują identyczne zbiory obserwowalnych zachowań, czyli $TBeh_{i+1} = TBeh_i$. Różnica między nimi polega na tym, że S_{i+1} wnosi dodatkowe szczegóły o wewnętrznej strukturze programu, przedstawiając go jako kolekcję komponentów o mniejszym poziomie abstrakcji, czyli S_{i+1} jest pewną dekompozycją S_i.

Można rozróżnić w zasadzie dwa stosowane sposoby przejścia od specyfikacji do implementacji.

Pierwsze polega na pisaniu specyfikacji i implementacji w odmiennych językach. W takich przypadkach językiem specyfikacji jest zwykle język logiki, a językiem implementacji – język wykonywalny. Poprawność implementacji zapewnia się przez udowodnienie zachodzenia odpowiednich związków między specyfikacją a implementacją.

Drugie podejście polega na transformacji specyfikacji. Jeżeli specyfikacja początkowa jest wykonywalna, ale – być może – mało efektywna, wtedy jej przekształcenie w drodze zabiegów składniowych, z którymi niekiedy wiąże się obowiązek dowodzenia poprawności ich stosowania, daje nową, bardziej efektywną specyfikację. Innym celem transformacji specyfikacji może być zmiana reprezentacji w postaci abstrakcyjnej na postać bliższą końcowej implementacji. Jeszcze innym celem może być uściślenie specyfikacji.

LITERATURA

[1] Baeten J.C.M., Bergstra J.A.: Real Space Process Algebra, W.: J.C.M. Baeten, J.F. Groote (eds): CONCUR'91, LNCS, Vol. 527, pp. 96–110, Springer-Verlag 1992

[2] van Benthem J.: Time, Logic and Computation, W: de Bakker J.W., de Roever W.P., Rozenberg G. (ed): Linear Time, Branching Time and Patial Order in Logics and Models for Concurrency, LNCS, Vol. 354, Springer-Verlag 1989

[3] Halang A.W., Sacha K.M.: Real-Time Systems – Implementation of Industrial Computerised Processes Automation. World Scientific, 1992

[4] Huzar Z., Spławski Z.: Tworzenie oprogramowania sieciowego – problemy metodologiczne i teoretyczne. Sieci Komputerowe'89, Prace Naukowe Centrum Obliczeniowego Politechniki Wrocławskiej 9, Konferencje 4, 132–148, Wydawnictwo Politechniki Wrocławskiej 1989

[5] Koymans R., Kuiper R., Zijstra E.: Paradigms for Real-Time Systems. W: Formal Techniques in Real-Time and Fault-Tolerant Systems, Joseph (ed), LNCS, Vol. 331, Springer-Verlag, pp. 159–174, 1988

[6] Lamport L., Lynch N.: Distributed Computing: Models and Methods. W: Handbook of Theoretical Computer Science, J. van Leeuwen (ed), Vol. 2, 1157–1199, Elsevier Science Publishers B.V., 1990

[7] Levi S-T. Agrawala A.K.: Real-Time System Design. Mc Graw-Hill Publishing Company, 1990

[8] Milner R.: Communication and Concurrency. Prentice Hall 1989

[9] Stankovic J.A., Ramamrithan K., (eds): Hard Real-Time Systems, IEEE Computer Society Press, 1988.

Systemy czasu rzeczywistego – metody specyfikacji

Zbigniew Huzar
Centrum Informatyczne
Politechnika Wrocławska

Jan Magott
Instytut Cybernetyki Technicznej
Politechnika Wrocławska

Metody specyfikacji można podzielić na dwie grupy: *deskryptywne i prospektywne*. Podstawą metody deskryptywnej jest wybór odpowiedniej logiki, czyli języka formuł wraz z odpowiednią interpretacją, mającego służyć do wyrażania interesujących własności systemu. Nieodłącznym elementem logiki jest system dowodzenia, który pozwala na wyprowadzenie z założonych własności nowych własności specyfikowanego systemu. Często logiki te są rozszerzane i ujmują systemy dowodzenia poprawności programów implementowanych w konkretnych językach programowania. Tego typu podejścia opierają się na odpowiednim rozszerzeniu metod specyfikacji i weryfikacji znanych dla klasycznego programowania współbieżnego [2], a odpowiednich przykładów dostarczają prace [4, 6]. Metody tej grupy opierają się na rozszerzeniach logiki klasycznej lub logiki temporalnej [8].

Podstawą metody prospektywnej jest mechanizm generowania dopuszczalnych obliczeń lub zachowań specyfikowanego systemu. Można w tej grupie metod wyróżnić dwie podgrupy, umownie nazywane jako: *językowa* i *automatowa*. Podejście językowe polega na opisie specyfikowanego systemu przez wyrażenie (odpowiednik programu), które może być generatorem zachowań systemu. Generowanie takie można realizować komputerowo. Przykładami podejść z tej podgrupy są języki LOTOS [3], ESTELLE i SDL oraz ich czasowe rozszerzenia. Podejście automatowe polega na przedstawieniu niektórych skończonych struktur wraz z operacjami na tych strukturach. Przykładami są tu struktury zdarzeniowe [11], sieci Petriego i ich czasowe rozszerzenia.

Tytułem ilustracji, w dalszej części artykułu, zostaną przedstawione dwa przykłady tylko samych języków logicznych, bez związanych z nimi systemów dowodzenia, oraz czasowe rozszerzenia sieci Petriego.

Logika RTL

Logika RTL stanowi proste rozszerzenie logiki klasycznej. Zaproponowana przez [7] była później rozszerzana w pracy [10], z której pochodzi używana niżej konwencja języka.

Punktem wyjścia jest przyjęcie następującego modelu specyfikowanego systemu czasu rzeczywistego. Podstawową jednostką składową systemu jest *akcja*. Akcja może być interpretowana jako operacja, której wykonanie wymaga pewnego czasu. Akcje można składać ze sobą sekwencyjnie i równolegle. Jedną z interpretacji dotyczących współpracy akcji jest przyjęcie komunikacji między akcjami przez synchroniczną wymianę komunikatów. Wykonanie akcji może rozpocząć się po uprzedniej aktywacji. Aktywacja akcji oznacza wskazanie, że akcja jest gotowa do wykonywania. Wykonywana akcja po jej zakończeniu może być ponownie aktywowana i wykonywana. Aktywowanie, rozpoczęcie i zakończenie akcji są reprezentowane przez odpowiednie zdarzenia. Akcja może być wykonywana wielokrotnie. Poza wymienionymi rodzajami zdarzeń możemy mieć jeszcze zdarzenia zachodzące poza systemem, czyli zdarzenia zewnętrzne.

Zdarzenia zachodzą natychmiastowo w jednej chwili czasowej, którymi są liczby naturalne. *Zajście* zdarzenia jest reprezentowane przez funkcję wystąpienia zdarzenia. Kolejne i-te zajście określonego zdarzenia jest reprezentowane przez funkcję wystąpienia $f(i)$; wartością tej funkcji jest chwila wystąpienia zdarzenia. Funkcje wystąpienia zdarzenia mają oczywiste własności, np. zdarzenie zakończenia akcji musi być poprzedzone zdarzeniem rozpoczęcia akcji, zdarzenie rozpoczęcia akcji – zdarzeniem aktywacji akcji.

Akcja w danym momencie jest charakteryzowana przez swój stan. Stan ten może ulegać zmianie w momencie spełnienia określonych warunków czasowych – od zajścia określonych zdarzeń. Warunki są reprezentowane przez predykaty czasowe, których wartości mogą ulegać zmianom w czasie pod wpływem rejestrowanych zdarzeń. Argumentami warunków są funkcje wystąpienia zdarzeń, krótko – zdarzenia.

W związku z wprowadzeniem pojęcia warunku zachodzi potrzeba wyróżnienia jeszcze dodatkowych rodzajów zdarzeń:

- Dwa zdarzenia zmiany stanu – określają moment, w którym dany czasowy warunek zmienił swoją wartość; istnieją dwa typy takich zdarzeń: jedno oznaczające zmianę wartości warunku z prawdy na fałsz oraz drugie – oznaczające zmianę z fałszu na prawdę; jeżeli warunek ma nazwę c, to pierwsza z tych funkcji będzie oznaczana *cff*, a druga – *ctf*;
- Trzy zdarzenia powiadomienia określają moment powiadomienia akcji o zajściu konkretnego zdarzenia; istnieją trzy typy takich zdarzeń: zdarzenia powiadomienia o tym, że nastąpiło zdarzenie zmiany określonego warunku na prawdę, zmiany na fałsz, lub zdarzenie zewnętrzne; jeżeli c jest nazwą warunku, to pierwsza z tych funkcji będzie oznaczana *cnt*, druga *cnf*, trzecia *ef*, gdzie e jest nazwą zdarzenia zewnętrznego.

Na podstawie przedstawionego modelu definiuje się język, który może służyć zarówno do specyfikacji wymagań, jak i specyfikacji projektu systemu. Należy dodać, że omawiany tu język służy do opisu tylko własności czasowych akcji, natomiast pomija własności odnoszące się do treści akcji, czyli do wartości przetwarzania danych. Odpowiednie rozszerzenie języka w celu uwzględnienia tych własności jest proste i tutaj pomijamy je. Celem zilustrowania użycia języka do sporządzenia specyfikacji wymagań oraz specyfikacji projektu systemu, rozpatrzmy prosty przykład systemu mającego jedno wejście i jedno wyjście. Na wejściu zachodzi zdarzenie zewnętrzne charakteryzowane przez funkcję zdarzeń f. O zdarzeniu tym wiadomo, że kolejne jego wystąpienia nie zachodzą częściej niż co p jednostek czasu. Własność ta wyraża własność otoczenia systemu i jest reprezentowana przez formułę:

$$\forall i.\, f(i) + p \leqslant f(i+1)$$

Od zdarzenia, które zachodzi na wyjściu, reprezentowanego przez funkcję zdarzeń g, żąda się tylko, aby kolejne jego wystąpienie następowało po zdarzeniu wejściowym. Żądanie to wyraża formuła:

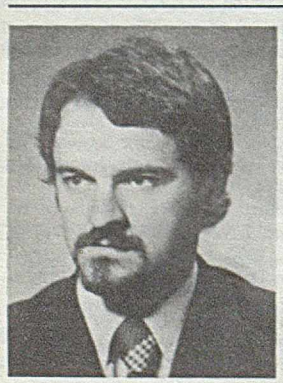
$$\forall i.\, f(i) < g(i)$$

Koniunkcja obu formuł, oznaczana $SpecReq(f, g)$, stanowi specyfikację wymagań dla systemu. Próbując zaprojektować system spełniający te wymagania moglibyśmy przyjąć, że system składa się z jednej akcji, której rozpoczęcie – zdarzenie s – jest powodowane zajściem zdarzenia f. Akcja ta kończy się – zdarzenie c po k jednostkach czasu od jej rozpoczęcia. Zdarzenie wyjściowe g ma zachodzić w trakcie wykonywania akcji. Wyraża to formuła:

$$SpecDes(f, g) = \forall i.\,(f(i) \leqslant s(i)) \wedge (s(i) \leqslant f(i) + r) \wedge (s(i) = c(i) - k) \wedge (s(i) < g(i) < c(i)),$$

gdzie k, r są stałymi. Formuła ta zawiera określone decyzje projektowe. Np. drugi z członów koniunkcji $s(i) \leqslant f(i) + r$ określa sposób szeregowania akcji do wykonania. Łącznie cała formuła musi spełniać ograniczenia wyrażone przez specyfikację wymagań. Wymóg ten można wyrazić formalnie w postaci implikacji:

$$SpecDes(f, g) \Rightarrow SpecReq(f, g)$$



Prof. dr hab. inż. JAN MAGOTT ukończył w 1975 r. studia na Wydziale Elektroniki Politechniki Wrocławskiej. W 1978 r. obronił pracę doktorską z zakresu teorii automatów. W 1989 r. otrzymał stopień dr. hab. za pracę „Ocena wydajności systemów komputerowych z zastosowaniem sieci Petriego". Był członkiem Komitetu Programowego Konferencji Petri Nets and Performance Models w: Kioto (1989), Melbourne (1991) oraz Tuluzie (1993). Obecnie pracuje na stanowisku profesora nadzwyczajnego w Instytucie Cybernetyki Technicznej Politechniki Wrocławskiej.

Jeżeli formuła ta jest spełniona, to oznacza poprawność specyfikacji projektowej względem specyfikacji wymagań.

Poniżej ograniczymy się do przedstawienia – jako prostszych – dwu przykładów specyfikacji wymagań. Przyjmiemy następującą konwencję nazw występujących w formułach: litera *f* na końcu symbolu oznacza funkcję (wystąpienia zdarzeń).

Żądanie reakcji. Dla danego warunku *c* żądamy, aby w przypadku zajścia zdarzenia zewnętrznego *e*, nastąpiła zmiana wartości tego warunku, o ile w momencie zajścia zdarzenia warunek ten był fałszywy:

$$\forall i_2 j.(cff(j) < ef(i) \wedge \neg\, \exists k.(cff(j) < ctf(k) < ef(i)) \Rightarrow \exists 1.(ctf(1) = ef(i))$$

Żądanie periodycznej aktywacji akcji przy założeniu, że prawdziwy jest dany warunek *c*. Oznaczenia: *aper* – stała określająca czas trwania akcji *a*; *aaf* – funkcja zdarzenia aktywacji akcji *a*:

$$\exists aper.\forall i,z.(cnf(i) < z^{*}aper \wedge \neg\, \exists j.(cnf(i) < cnf(j) < z^{*}aper \Rightarrow \exists k.(aaf(k) = z^{*}aper)$$

Logika temporalna

Tradycyjnie, logika temporalna stosowana do specyfikowania i analizy systemów reaktywnych [9], może być dogodnym narzędziem do specyfikowania systemów czasu rzeczywistego przy jakościowym traktowaniu czasu. Podejście takie ma określone zalety: znacznie upraszcza się analiza i wnioskowanie, a dodatkowo wyniki analizy są silne w tym sensie, że tego typu własności mogą być również spełniane przez szeroką klasę implementacji takich systemów. Jednak w wielu przypadkach nie można uciec od ilościowego traktowania czasu i dlatego istnieją różne propozycje odpowiedniego rozszerzenia logiki temporalnej.

Poniżej zostanie przedstawiona jedna z takich propozycji [15] nazwana GCTL (*Global Clock Temporal Logic*). Nazwa logiki wynika z przyjętego modelu upływu czasu. Zakłada się model czasu globalnego – bieżący czas jest reprezentowany przez globalną zmienną *T*, która przyjmuje wartości chwil czasowych należące do dowolnie ustalonego gęstego i liniowo uporządkowanego zbioru. Zbiorem takim może być np. zbiór liczb wymiernych.

Przedstawiana logika jest ukierunkowana na specyfikowanie obliczeń systemu, rozumianych jako ciąg stanów. Ukierunkowanie logiki na wyrażanie własności ciągów stanów nie jest ograniczeniem nie pozwalającym na wyrażanie własności ciągów interakcji towarzyszących przechodzeniu między stanami. Ciągi interakcji można reprezentować jako element stanów przez wprowadzenie pomocniczych zmiennych historycznych.

Wyróżnia się dwie kategorie formuł logiki: formuły odnoszące się do stanu i formuły odnoszące się do obliczeń. Pierwsza kategoria formuł, to formuły logiki klasycznej nazywane formułami stanowymi, natomiast drugą kategorią są formuły temporalne. Przyjmując, że Czytelnikowi jest znana logika klasyczna, najpierw krótko przypominamy język logiki temporalnej bez czasu rzeczywistego, a następnie język jej rozszerzenia czasowego.

Formuła temporalna składa się z formuł klasycznych i operatorów temporalnych. Operatorami tymi są: unarny operator ***next*** oraz binarny ***until***. Operatory te można wykorzystać do definiowania innych operatorów temporalnych, np. unarny operator ***Sometime*** Φ, służący do wyrażania, że w pewnej określonej przyszłej chwili zostanie osiągnięty stan, w którym będzie spełniona formuła Φ, jest zdefiniowany jako *true* ***until*** Φ, gdzie *true* oznacza formułę tożsamościowo prawdziwą. Niech obliczenie σ będzie ciągiem:

$$(s_{(}, t_{(}), (s_1, t_1), \ldots$$

Para (s_j, t_j) obliczenia σ będzie oznaczana przez (σ_2, j). Jeżeli obliczenie σ jest skończone, to $|\sigma|$ oznacza jego długość. Spełnialność formuły temporalnej Φ w stanie (σ_2, j) danego obliczenia σ jest definiowana indukcyjnie:

$(\sigma_2 j)| = \neg\, \Phi \quad \Leftrightarrow (\sigma_2 j)| \neq \Phi$

$(\sigma_2 j)| = \Phi \vee \Psi \quad \Leftrightarrow (\sigma_2 j)| = \Phi$ lub $(\sigma_2 j)| = \Psi$

$(\sigma_2 j)| = \textbf{next}\,\Phi \quad \Leftrightarrow (\sigma_2 j + l)| = \Phi$

$(\sigma_2 j)| = \Phi$ ***until*** $\Psi \Leftrightarrow \exists l.((j < l < |\sigma| \wedge (\sigma_2 j)| = \Psi)$ oraz $\forall i.j \leqslant i < l, (\sigma, i)| = \Phi)$

Rozszerzenie logiki temporalnej polega na dopuszczeniu do użycia w formułach wolnej zmiennej *T*, modelującej upływ czasu. Zmienna *T* zmienia swą wartość zgodnie z upływem czasu, przy czym w momentach zmiany stanu, zmiany te mogą być skokowe. Spełnialność formuły stanowej Φ w danym stanie (σ_2, j) obliczenia σ odnosi się do pary (s_j, t_j). Pociąga to zmianę interpretacji formuł stanowych: obliczenie wartości formuły stanowej Φ zamiast tylko do s_i odnosi się do (s_i, t_i). Np. formuła:

$$(\Phi \wedge T = t) \Rightarrow \textbf{sometime}\ (\Psi \wedge T \leqslant t + 5 \wedge T \geqslant t + 2)$$

stwierdza, że jeżeli Φ jest aktualnie prawdziwe (aktualną chwilę kojarzymy z wartością zmiennej *t*), to formuła musi stać się prawdziwa przynajmniej w jednej chwili odległej od chwili bieżącej co najmniej o dwie, a najwyżej o pięć jednostek czasu. Inna formuła:

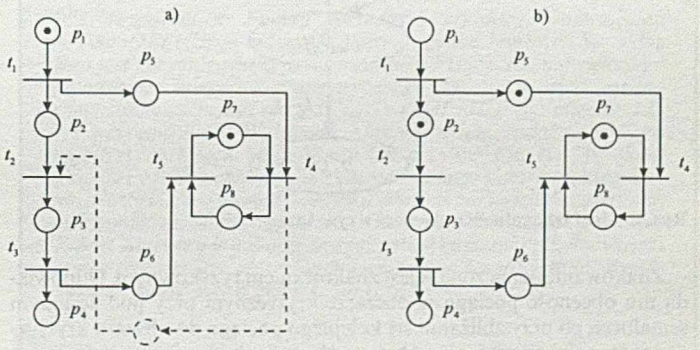
$$T = t \Rightarrow \textbf{next}\ (\Phi \wedge T \geqslant t + 2)$$

stwierdza, że Φ jest prawdziwe w następnym stanie obliczenia i system nie może być w tym stanie przed upływem trzech jednostek czasu.

Sieci Petriego

$N = \{0, 1, \ldots\}$ jest zbiorem liczb naturalnych. Sieć Petriego (*SP*) [14] jest czwórką uporządkowaną $N = \langle P, T, F, W\rangle$ gdzie $P = \{p_1, \ldots, p_m\}$ – zbiór miejsc, $T = \{t_1, \ldots, t_n\}$ – zbiór przejść, $F \subset (P \times T) \cup (T \times P)$ – zbiór łuków, $W: F \to N \setminus \{0\}$ – funkcja krotności łuku.

W reprezentacji graficznej (rys. 1), miejsca są przedstawiane w postaci kółek, natomiast przejścia – w postaci kresek. Jeśli krotność łuku $f \in F$ spełnia zależność $W(f) > 1$, to łuk jest opisany liczbą $W(f) = 1$. Jeśli $W(f) = 1$, to symbol 1 jest pomijany. *Znakowanie SP jest funkcją* $M: P \to N$. Wartość $M(p_i)$ wyraża liczbę znaczników miejsca p_i. Liczba znaczników jest oznaczana na rysunkach przez liczbę kropek w miejscu. *Znakowana sieć Petriego* (*ZSP*) jest parą $M = \langle N, M_0\rangle$, gdzie M jest znakowaniem sieci Petriego N, zwanym znakowaniem początkowym. Zbiór $t = \{p \in P: \langle p, t\rangle \in F\}$ ($t = \{p \in P: \langle t, p\rangle \in F\}$) jest *zbiorem miejsc wejściowych* (*wyjściowych*) przejścia *t*. *Przejście* t_j *jest przygotowane do* odpalenia, jeśli w każdym miejscu $p_i \in t_j$ znajduje się co najmniej $W(\langle p_1, t_j\rangle)$ znaczników jest usuwanych z każdego miejsca $p_1 \in t_j$ oraz $W(\langle t_j, p_k\rangle)$ jest dodawanych do każdego miejsca $p_k \in t_j$.



Rys. 1. Znakowana sieć Petriego

Jako przykład rozważmy sieć z rys. 1. Sieć ta obrazuje proces ruchu pociągu przez obszar krytyczny, w którym może być co najwyżej jeden pociąg. Ruch przez obszar krytyczny jest sterowany za pomocą semafora. Znaczenie poszczególnych miejsc jest następujące:

p_1 – pociąg zbliża się do obszaru krytycznego,
p_2 – pociąg jest blisko obszaru krytycznego,
p_3 – pociąg jest w obszarze krytycznym,
p_4 – pociąg jest za obszarem krytycznym,
p_7 – semafor jest podniesiony,
p_8 – semafor jest opuszczony.

Przejście t_4 modeluje opuszczenie semafora, natomiast przejście t_5 – podniesienie semafora. Odpalenie przejścia t_1 generuje sygnał potrzeby opuszczenia semafora, natomiast odpalenie przejścia t_3 – sygnał potrzeby podniesienia semafora.

Dla znakowania sieci, ilustrowanego na rys. 1a, może być odpalone tylko jedno przejście t_1. Po odpaleniu tego przejścia jest osiągane znakowanie wyrażone rys. 1b. Jeśli w wyniku odpalenia przejścia t_i następuje zmiana znakowania z M_i na M_{i+1}, to stosujemy oznaczenie $M_i \xrightarrow{t_i} M_{i+1}$. *Sekwencją odpaleń* nazywamy taką sekwencję przejść, dla której istnieją znakowania $M_p, M_{p+1}, \ldots, M_{p+s}$ spełniające warunek

$$M_p \xrightarrow{t_{i1}} M_{p+1}, \ldots, M_{p+s-1} \xrightarrow{t_{is}} M_{p+s}$$

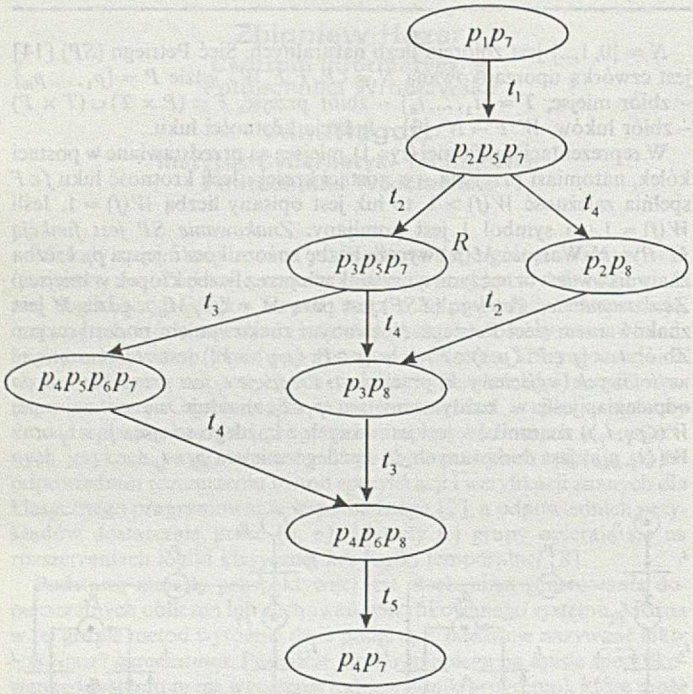
W powyższym przypadku stosujemy oznaczenie $M_p \xrightarrow{\sigma} M_{p+s}$. Znakowanie M_{p+s} nazywamy znakowaniem osiągalnym ze znakowania M_p.

Zbiór znakowań osiągalnych ze znakowania początkowego M_0 jest zdefiniowany następująco:

$$R(M_0) = \{M : \sigma \in T^* \wedge M_0 \xrightarrow{\sigma} M\}$$

gdzie T^* jest zbiorem wszystkich sekwencji skończonej długości ze zbioru przejść.

Dla sieci o skończonym zbiorze znakowań osiągalnych ze znakowania początkowego M_0, zmiany znakowań sieci przez odpalanie przejść obrazuje *graf osiągalności*. Rys. 2 obrazuje graf osiągalności dla sieci z rys. 1a. Wierzchołki tego grafu są znakowaniami, przy czym zbiór miejsc umieszczonych w elipsie wyraża zbiór miejsc, w których występują znaczniki dla danego znakowania.



Rys. 2. Graf osiągalności dla sieci z rys. 1a

Znakowanie oznaczone R jest znakowaniem ryzykownym. Odpowiada mu obecność pociągu w obszarze krytycznym przy podniesionym semaforze, co przy zbliżaniu się kolejnego pociągu do obszaru krytycznego, stanowi zagrożenie. Aby zapobiec znakowaniu ryzykownemu, w pierwszej kolejności powinno być odpalane przejście t_4 (modelujące opuszczenie semafora), a dopiero potem – przejście t_2 (wyrażające wjazd pociągu w obszar krytyczny). W tym celu sieć Petriego z rys. 1a rozszerzamy o fragment oznaczony linią przerywaną.

Czasowe sieci Petriego

Czasową siecią Petriego [16] jest para $M' = \langle M, T\rangle$, gdzie M jest ZSP, $T: T \to R'_+$ (R'_+ jest zbiorem nieujemnych liczb wymiernych) jest funkcją czasu odpalania przejść. Gdy przejście t_i jest przygotowane do odpalenia, wówczas może nastąpić jego odpalenie. W chwili rozpoczęcia odpalania przejścia t_i, z każdego z miejsc wejściowych p_j przejścia t_i usuwanych jest $W(\langle p_j, t_i\rangle)$ znaczników. Odpalanie przejścia t_i trwa przez czas $T(t_i)$. W chwili zakończenia odpalania przejścia t_i do każdego z miejsc wyjściowych p_k tego przejścia dodawanych jest $W(\langle t_i, p_k\rangle)$ znaczników.

Ponownie rozważmy sieć Petriego z rys. 1a bez części oznaczonej linią przerywaną. Czas odpalania części t_2 modeluje czas od chwili przejazdu pociągu przez wyróżniony punkt, w którym jest wysyłany sygnał potrzeby opuszczenia semafora, do chwili wjazdu w obszar krytyczny. Czas odpalania przejścia t_4 wyraża czas od chwili przejazdu pociągu przez wymieniony wyróżniony punkt do chwili opuszczenia semafora. Jeśli czasy odpalania przejść t_2, t_4 spełniają relację

$$T(t_4) < T(t_2),$$

to znakowanie ryzykowne nie zostanie osiągnięte.

Rozważmy zagadnienie adekwatności powyższego modelu. Czas przejazdu pociągu między rozważanymi punktami nie jest wielkością równą dla wszystkich przejazdów między tymi punktami. Stąd czas odpalania przejścia t_2, zadany jedną liczbą rzeczywistą, nie jest adekwatnym modelem.

Sieci Petriego z mechanizmem przeterminowania

W sieciach Petriego z mechanizmem przeterminowania [12], czasy odpalania przejść są zadane za pomocą funkcji minimalnego czasu odpalania przejść *Min* oraz funkcji maksymalnego czasu odpalania przejść *Max*. Aby w wyżej rozważanym przypadku uniknąć stanu ryzykownego, funkcje minimalnego i maksymalnego czasu odpalania przejść powinny spełniać warunek:

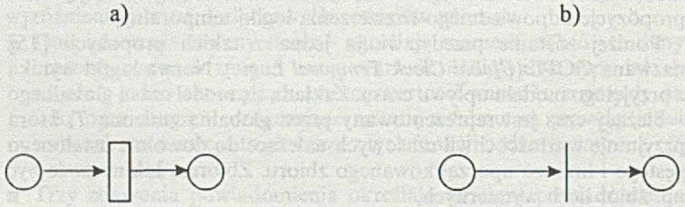
$$Max(t_4) < Min(t_2)$$

$Min(t_2)$ jest minimalnym czasem przejazdu pociągu od punktu wysyłania sygnału potrzeby opuszczenia semafora, do punktu wjazdu w obszar krytyczny. Dla uzyskania odpowiednio dużej wartości $Min(t_2)$ można narzucić ograniczenie prędkości przed obszarem krytycznym lub zwiększyć odległość między dwoma rozważanymi punktami.

Stochastyczne sieci Petriego

Poniżej przedstawimy *uogólnione stochastyczne sieci Petriego* (USSP) [1] z racji najwyższej ich rangi wśród stochastycznych sieci Petriego. USSP zawiera następujące typy przejść:

- przejścia czasowe – o czasie odpalania opisanym zmienną losową o rozkładzie wykładniczym (rys. 3a),
- przejścia natychmiastowe – o zerowym czasie odpalania (rys. 3b).



Rys. 3. Przejścia: a) czasowe, b) natychmiastowe

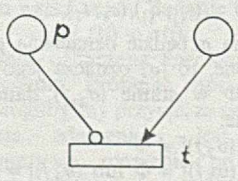
Przejścia natychmiastowe mają priorytet nad przejściami czasowymi w tym sensie, że jeśli do odpalenia przygotowane jest przynajmniej jedno przejście natychmiastowe, to niezależnie od przygotowanych przejść czasowych – w pierwszej kolejności zostanie odpalone przejście natychmiastowe.

USSP jest siódemką:

$$USSP = \langle P, T, P, F, H, M_0, G\rangle$$

gdzie P, T, F, M_0 mają znaczenie analogiczne, jak w podstawowej definicji sieci Petriego, przy czym łuki są łukami jednokrotnymi;
$H \subset P \times T$ jest zbiorem łuków wzbraniających, łuki wzbraniające zakończone są małym kółkiem (rys. 4); jeśli w miejscu p na rys. 4 jest znacznik, to przejście t nie może być odpalone;
$P: T \to N$ jest funkcją priorytetu;
$P(t) = 0$ dla przejścia czasowego (najniższy priorytet),
$P(t) \geqslant 1$ dla przejść natychmiastowych,
$G = \langle g_1, g_2, \ldots, g_n\rangle$ jest wektorem, którego składowa g, jest:

- parametrem λ, rozkładu wykładniczego, jeśli przejście t_i jest czasowe,
- wagą (stosowaną przy obliczaniu prawdopodobieństw odpalenia przejść natychmiastowych), jeśli t_i jest natychmiastowe,



Rys. 4. Przejście z łukiem wzbraniającym

Jeśli przygotowane jest do odpalenia więcej niż jedno przejście natychmiastowe, to o wyborze przejść do odpalenia w pierwszej kolejności rozstrzyga priorytet. Jeśli zbiór przygotowanych do odpalenia przejść o najwyższym priorytecie zawiera więcej niż jeden element, to przejścia do odpalenia są wybierane z prawdopodobieństwami zależnymi od wag. Prawdopodobieństwo wyboru przejścia do odpalenia jest proporcjonalne do jego wagi.

USSP można stosować do wyznaczania np. prawdopodobieństwa przekroczenia ograniczenia czasowego [13].

* * *

Powyżej tylko szkicowo przedstawiono metody specyfikacji. Przegląd ten nie daje podstaw do oceny i określenia zakresu ich zastosowania. Pominięto szczegóły dotyczące m.in. przyjmowania modeli obliczeń systemów czasu rzeczywistego, możliwości strukturalizacji opisów, systemów dowodzenia dla metod deskryptywnych i semantyki operacyjnej dla metod prospektywnych.

Moc opisowa RTL nie wykracza poza logikę klasyczną, natomiast logika temporalna jest systemem szerszym niż logika klasyczna. Stąd logika temporalna jest systemem szerszym niż RTL.

Przedstawione modele sieci Petriego nie zawierają predykatów, których reprezentacja jest możliwa w systemach logicznych, takich jak RTL czy logika temporalna. Istnieje klasa sieci Petriego wysokiego poziomu, tzw. dobrze zbudowanych, kolorowanych, stochastycznych sieci Petriego, w których można wyrażać ograniczoną klasę predykatów.

RTL i logika temporalna dopuszczają niedeterminizm, ale w żadnym z tych formalizmów nie ma możliwości określania takich miar niedeterminizmu, jak rozkłady prawdopodobieństwa.

Teraz krótko scharakteryzujemy narzędzia programistyczne do badania powyższych modeli. Należy podkreślić, że tego rodzaju narzędzia nie są jeszcze zbyt doskonałe. Narzędzia specyfikacji występują często jako element konkretnych środowisk programistycznych (np. dla języka Ada) bądź jako element narzędzi typu CASE (np. system EPOS [5]. Są to zwykłe narzędzia nie dysponujące pełnymi własnościami metod formalnych.

Budowa narzędzi wspomagających tworzenie i badanie własności specyfikacji deskryptycznych napotyka te same problemy, jakie są znane z prac nad automatyzacją systemów dowodzenia własności programów sekwencyjnych i współbieżnych [2]. Dlatego większość proponowanych narzędzi ma charakter studialny. Na uwagę zasługuje podejście specyfikacji logicznej (język specyfikacji FRORL [17]) oparte na logice klauzul Horna, dla której istnieją efektywne metody dowodzenia.

Bardziej zaawansowane są prace nad tworzeniem narzędzi związanych z metodami prospektywnymi. Dla sieci Petriego z czynnikami czasu istnieje już kilka pakietów, spośród których najbardziej rozpowszechniony jest pakiet *Great GSPN*, przeznaczony do analizy uogólnionych stochastycznych sieci Petriego. Sporo narzędzi programowych wiąże się z formalnymi technikami specyfikacji LOTOS oraz ESTELLE, rozwijanymi przez ISO. Obecne wersje tych narzędzi uwzględniają czas rzeczywisty tylko częściowo, natomiast można spodziewać się, że ich rozwój pójdzie właśnie w kierunku czasu rzeczywistego.

LITERATURA

[1] Ajmone Marsan M., Balbo G., Conte G.: A Class of Generelized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems. ACM Trans. Comp. Syst., Vol 2, pp. 93–122, 1984

[2] Apt K.R., Olderog E.R.: Verification of Sequential and Concurrent Programs. Springer-Verlag 1991

[3] van Eijk P.H.J., Vissers C.A., Diaz M. (eds): The Formal Description Technique LOTOS, North-Holland 1989

[4] Haase V.H.: Real-Time Behaviour of Programs. IEEE Trans. on Software Engineering, Vol. SE-7, pp 494–501, 1981

[5] Halang A.W., Sacha K.M.: Real-Time Systems – Implementation of Industrial Computerised Processes Automation. World Scientific, 1992

[6] Huzar Z.: Programowanie procesów komunikujących się w czasie rzeczywistym. Prace Naukowe Centrum Obliczeniowego Politechniki Wrocławskiej 8, Monografie 1, Wydawnictwo Politechniki Wrocławskiej, 1989

[7] Jahanian F., Mok A.K.: Safety Analysis of Timing Properties in Real-Time Systems. IEEE Trans. on Software Engineering, Vol SE-12, pp. 890–904, 1986

[8] Joseph (ed): Formal Techniques in Real-Time and Fault-Tolerant Systems, LNCS vol 331, Springer-Verlag, 1988

[9] Kröger F.: Temporal Logic of Programs. EATCS Monographs on Computer Science 8, Springer-Verlag, 1987

dokończenie na s. 30

Prosto z firm ● Prosto z firm

Microsoft i NTT (Nippon Telegraph and Telephone Corporation) podpisały porozumienie w celu wspólnego opracowania usług dla dostarczania informacji multimedialnych przez sieci telekomunikacyjne w Japonii. W ramach porozumienia wykorzystana będzie technologia sieciowa NTT oraz ekspertyza Microsoft w dziedzinie systemów operacyjnych. Na początku informacje multimedialne będą wymieniane za pośrednictwem CD-ROM, sieci analogowych i cyfrowych. Użytkownicy będą mogli np. tanio kupić dyskietkę CD-ROM, przejrzeć jej zawartość, a następnie przez sieć uzyskać klucz uprawniający do wykorzystania zawartości CD-ROM. Obie firmy już współpracują nad projektem mającym na celu uzyskanie zgodności między systemem Microsoft At Work a F-NET – pakietem usług faksowych NTT.

* * *

Podczas spotkania Europejskiej Grupy Użytkowników Oracle w Maastrich firma Oracle zaprezentowała pierwszy zestaw otwartych aplikacji klient-serwer, automatyzujących decydujące w działalności przedsiębiorstwa operacje. Oracle Cooperative Applications Realase 10 to zestaw oparty na technologiach Oracle7, Oracle Parallel Server i Oracle CDE, umożliwiający przetwarzanie równoległe i w środowisku rozproszonym. W skład zestawu wchodzi ponad 25 indywidualnych modułów wspomagających księgowość, produkcję, dystrybucję, zarządzanie zasobami ludzkimi i projektami. Są one dostępne na wszystkich podstawowych platformach sprzętowych, systemach operacyjnych i protokołach sieciowych.

* * *

ICL rozpoczął w Japonii sprzedaż brytyjskiego rozwiązania – TeamOFFICE – automatyzującego prace biurowe. Sprzedaż jego jest zorganizowana przez struktury FUJITSU, największego japońskiego koncernu informatycznego, który jak wiadomo ma większościowy pakiet akcji ICL. TeamOFFICE ma obecnie 125 tys. użytkowników. W Japonii – jak wykazują sondaże – w ciągu najbliższych trzech lat powinno ich przybyć 150 tys. W skład TeamOFFICE wchodzą następujące moduły: poczta elektroniczna, archiwizacja i współdziele dokumentów, zarządzanie zapasami, elektroniczna tablica ogłoszeń i kontrola obiegu dokumentów. Pakiet zapewnia integrację ze znanymi arkuszami kalkulacyjnymi oraz z powszechnie stosowanymi edytorami tekstów, m.in. WordPerfect, Word for Windows, AmiPro.

* * *

IBM i Hitachi podpisały porozumienie technologiczne i licencyjne dotyczące komputerów centralnych opartych na technologii CMOS oraz równoległych systemów RISC z architekturą POWER/Power PC. IBM będzie w przyszłości dostarczał firmie Hitachi nowe wersje mikroprocesorów S/30 CMOS, przystosowanych do specyfiki Hitachi, celem ich integracji w systemy, których produkcja zacznie się za 2 lata. Hitachi wykorzysta też architekturę IBM POWER/Power PC w swoich najnowszych systemach typu RISC.

* * *

PolSpark wprowadził do sprzedaży nowe modele komputerów z serii HETMAN 10 – wyposażone w płyty główne typu SPARCStation(R)10+ firmy Sun. Nowe modele charakteryzują się taką samą wydajnością jak komputery SPARCStation 20 zaprezentowane niedawno przez Sun Microsystems. Komputery PolSPARK pracują z modułami SuperSPARC o częstotliwościach zegara 40, 50 i 60 MHz, zaś wewnętrzne magistrale MBus i SBus pracują z częstotliwościami 50 MHz i 25 MHz.

* * *

Microsoft rozpoczął sprzedaż programu dla dzieci – Fine Artist, który umożliwia tworzenie projektów malarskich, rysunkowych i multimedialnych. Jest to uzupełnienie dostępnego już od ub.r. produktu o nazwie Creative Writer. Program zawiera zestaw narzędzi do samodzielnego tworzenia przez dzieci plakatów, animowanych obrazów, nalepek, pokazów multimedialnych itp. Do uruchomienia programu potrzebny jest komputer z procesorem 386 lub lepszym, Windows 3.1, 4 MB pamięci RAM, karta VGA, 11 MB miejsca na dysku; zalecane jest posiadanie karty dźwiękowej.

Prosto z firm ● Prosto z firm

Standardy planowania jakości, dokumentacji i konserwacji oprogramowania

Krzysztof Sacha
Instytut Informatyki
Politechnika Warszawska

Typowy cykl oprogramowania, w szczególności oprogramowania systemu sterującego, można umownie podzielić na kilka etapów, w których powstają kolejne elementy dokumentacji projektowej, testy i ostatecznie – gotowy system. Osiągnięcie wysokiej jakości oprogramowania przy rozsądnym poziomie kosztów, wymaga wielu przedsięwzięć organizacyjnych, normujących sposób prowadzenia prac i kontroli wyników każdego etapu. Podstawową rolę odgrywa w tym zakresie dokumentacja projektowa oraz związany z nią system weryfikacji i oceny.

Zwarty opis organizacji cyklu opracowania oprogramowania, uzupełniony krótkim przeglądem metod projektowych można znaleźć w [1]. Treścią tego artykułu jest zwięzły przegląd standardów ANSI/IEEE, określający zasady:

- planowania i kontroli jakości oprogramowania [3, 4, 5, 6],
- dokumentowania projektów programowych [7, 8, 9],
- organizacji procesu konserwacji i atestacji zmian [10, 11].

Tabela 1. Struktura kosztów opracowania oprogramowania

Nazwa etapu	Oszacowanie kosztu w %
Specyfikacja wymagań	15
Projekt wstępny	7
Projekt szczegółowy	18
Programowanie modułów	20
Integracja i testowanie komponentów	20
Instalacja i ocena	20
Konserwacja	**70 ... 200%**

Oryginalny projekt = 100%

Każde z omówionych zagadnień zostanie omówione w odrębnym rozdziale. Ostatni rozdział, dotyczący wprowadzania i atestacji zmian, uwzględnia również zalecenia komitetu EWICS TC7 [2], dotyczące integralności systemów sterujących instalacjami potencjalnie niebezpiecznymi. Artykuł nie obejmuje obszernej tematyki standaryzacji procesu testowania.

Kontrola jakości oprogramowania

Jakość (ang. *quality*) oprogramowania definiuje się jako stopień, w jakim program, jego składnik lub proces, spełnia wymagania jakościowe. Osiągnięcie wysokiej jakości wymaga nieustannej kontroli, czy wyniki cząstkowe, osiągane w kolejnych etapach procesu projektowego, są zgodne z wymaganiami określonymi na poprzednim etapie. Czynności kontrolne nie mogą być podejmowane *ad hoc*, lecz powinny być zaplanowane i określone przed rozpoczęciem projektowania, a więc we wczesnym etapie specyfikacji wymagań. Szczegółowy i systematyczny schemat postępowania w tym zakresie nosi nazwę **planu kontroli jakości** (*Software Quality Assurance Plan*).

Plan kontroli jakości jest dokumentem opisującym sposób prowadzenia projektu, działania kontrolne oraz osoby odpowiedzialne za wykonanie poszczególnych działań. Najważniejsze rozdziały planu powinny określać:

- wszystkie etapy projektu i powstające na każdym etapie dokumenty,
- przeglądy i rewizje przeprowadzane po zakończeniu kolejnych etapów,
- używane standardy, metody i narzędzia wspomagające,
- zasady przechowywania programów i dokumentacji oraz nadzoru nad spójnością wersji, a także procedurę wprowadzania zmian.

Etapy projektu i powstające w nich dokumenty. Zalecana sekwencja prac jest zgodna z podaną w tabeli 1. Minimalny zbiór dokumentów, jaki powinien zostać wytworzony podaje tabela 2.

Analiza i specyfikacja wymagań precyzuje warunki, jakie musi spełniać planowane oprogramowanie. Treść specyfikacji powinna określać wszystkie wymagane funkcje oprogramowania, jego wydajność i szybkość oraz sprzęg z otoczeniem. Każde z wymagań powinno być opisane w sposób umożliwiający obiektywną weryfikację i ocenę jego spełnienia.

Opis projektu wstępnego, definiujący sposób podziału oprogramowania na mniejsze jednostki, do poziomu jednostek programowych. Treść opisu powinna określać reguły współpracy tych jednostek i pokazywać, w jaki sposób zaprojektowana struktura spełnia wymagania podane w specyfikacji.

Opis projektu szczegółowego, określający wewnętrzną strukturę, dane i algorytmy działania wszystkich jednostek projektowych.

Dokumentacja użytkowa, opisująca sposób obsługi programów i wymagane dane wejściowe, wszystkie komunikaty błędów i wymagane akcje korekcyjne, a także sposób zgłaszania błędów wykrytych przez użytkowników.

Plan weryfikacji i oceny, określający metody sprawdzenia, że specyfikacja wymagań została zatwierdzona przez odpowiednie organy decyzyjne, że projekt spełnia wymagania zawarte w jej specyfikacji, oraz że kod programu realizuje funkcje ustalone w opisie projektu szczegółowego, a także określający metody oceny, że oprogramowanie spełnia wymagania zawarte w specyfikacji. Przykładowe metody weryfikacji i oceny obejmują przeglądy techniczne, inspekcje projektu i kodu, testowanie lub dowodzenie poprawności programu. Plan powinien też precyzować kryteria akceptacji produktu oraz terminarz testów.

Raport weryfikacji i oceny, opisujący rezultaty działań wymienionych w planie weryfikacji i oceny. Elementem raportu jest końcowy wniosek, określający czy oprogramowanie może być dopuszczone do eksploatacji.

Tabela 2. Podstawowe dokumenty i przeglądy

Nazwa etapu	Produkty	Przeglądy i rewizje
Analiza i specyfikacja wymagań	Specyfikacja wymagań Plan kontroli jakości Plan weryfikacji i oceny	Przegląd specyfikacji Przegląd planu kontroli jakości Przegląd planu weryfikacji i oceny
Projekt wstępny	Opis projektu wstępnego	Przegląd projektu wstępnego
Projekt szczegółowy	Opis projektu szczegółowego Dokumentacja użytkowa	Przegląd projektu szczegółowego Przegląd dokumentacji użytkowej
Implementacja	Kod programu z dokumentacją	
Integracja i testowanie	Raporty testowania	Rewizja funkcjonalna
Ocena	Całość oprogramowania i dokumentacji Raport weryfikacji i oceny	Rewizja fizyczna

Przeglądy i rewizje. Przeglądem (ang. *review*) nazywa się proces przedstawienia produktu projektantom, kierownictwu, użytkownikom, klientom lub innym zainteresowanym, w celu uzyskania ich opinii i oceny. Rewizją (ang. *audit*) nazywa się ocenę produktu, dokonywaną przez niezależny zespół, w celu stwierdzenia zgodności ze specyfikacją wymagań lub innymi wymaganiami kontraktowymi. Plan kontroli jakości powinien określać harmonogram przeglądów i rewizji, jakie muszą być przeprowadzone podczas trwania projektu,

oraz ustalać kryteria zatwierdzania lub odrzucania ocenianego produktu.

Minimalny zbiór przeglądów i rewizji (tab. 2) obejmuje:

Przegląd specyfikacji wymagań, sprawdzający adekwatność, wykonalność i kompletność sprecyzowanych wymagań. Plan kontroli jakości powinien określać przedmiot i takie kryteria oceny, jak: kompletność wykazu dokumentów związanych, kompletność i poprawność algorytmów i równań, zgodność z innymi dokumentami (sprzęgi sprzętowe i programowe), adekwatność opisu sprzęgu z użytkownikiem, weryfikalność (testowalność) wymagań oraz jednoznaczność, spójność i modyfikalność dokumentu.

Przegląd planu weryfikacji i oceny, sprawdzający adekwatność, kompletność oraz wykonalność metod weryfikacji i oceny zawartych w planie.

Przegląd projektu wstępnego, oceniający zgodność projektu wstępnego ze specyfikacją wymagań, jego wykonalność i techniczną poprawność.

Przegląd projektu szczegółowego, sprawdzający czy zaprojektowana struktura oprogramowania spełni wymagania zawarte w specyfikacji wymagań. Kryteria oceny opisu projektu są podobne do kryteriów oceny tej specyfikacji.

Rewizja funkcjonalna, przeprowadzana dla stwierdzenia, czy wszystkie wymagania zostały spełnione.

Rewizja fizyczna, przeprowadzona dla stwierdzenia, czy oprogramowanie i dokumentacja są wewnętrznie spójne i gotowe do użycia.

Dodatkowo jest zalecane przeprowadzenie w toku projektu wyrywkowych kontroli spójności wybranych fragmentów projektu oraz przeglądów zarządzania, oceniających stopień wykonania działań przewidzianych w planie kontroli jakości. Po zakończeniu projektu powinien odbyć się przegląd końcowy, służący podsumowaniu wykonanych działań i wyciągnięciu wniosków na przyszłość.

Standardy, konwencje i narzędzia. Plan kontroli jakości powinien narzucić projektantom stosowanie określonego formatu dokumentów, nazewnictwa, metod analizy wymagań i projektowania, a także konwencję umieszczania komentarzy oraz metody i miary testowania. Jeżeli jest to możliwe, plan powinien określać używane narzędzia, np. edytory tekstów, systemy wspomagające projektowanie, środowiska programowe, testery i symulatory.

Jako przykłady miar, określających poziom wiarygodności testowania można podać stosunek liczby przetestowanych do liczby wszystkich: rozejść warunkowych programu, gałęzi programu, komunikatów o błędach, indywidualnie wyodrębnionych wymagań itp.

Zgłaszanie i rozwiązanie problemów. Wynikające podczas eksploatacji oprogramowania problemy powinny być rozwiązywane w ustalony i systematyczny sposób. Plan kontroli jakości powinien określić obowiązującą procedurę zgłaszania, rejestracji i rozwiązywania problemów oraz ustalić odpowiedzialne za realizację tej procedury komórki organizacyjne.

Nadzór nad programami. Modyfikacja oprogramowania prowadzi do powstania wielu wersji tego samego produktu. Plan kontroli jakości powinien ustalać metody oraz środki stosowane do przechowywania, dokumentowania i zabezpieczania spójności kolejnych wersji. Związanym problemem jest określenie sposobów składowania programów, nanoszenia zmian, tworzenia kopii rezerwowych oraz ochrony programów przed nieuprawnionym dostępem i przypadkowym uszkodzeniem lub zniszczeniem.

Dokumentacja projektowa

Podstawowymi dokumentami określającymi techniczną zawartość projektu są: specyfikacja wymagań oraz opis projektu. Pierwszy dokument powinien dokładnie określić, co projektowany system ma realizować, jednak bez podawania sposobu wykonania. Drugi, powinien stwierdzać, jak projektowany system będzie spełniał wymienione w specyfikacji wymagania.

Specyfikacja wymagań (ang. *software requirements specification*) jest przygotowywana przez analityków badających potrzeby użytkowników oraz możliwości zaspokojenia tych potrzeb. Po zatwierdzeniu, specyfikacja wymagań staje się częścią umowy (kontraktu) między zleceniodawcą a wykonawcą prac projektowych oraz stanowi podstawę do wykonania i weryfikacji wszystkich dalszych prac. Błędy popełnione w specyfikacji przenoszą się automatycznie do dalszych etapów i z reguły są niewidoczne aż do ostatecznej oceny i prób eksploatacyjnych. Usunięcie tych błędów może być bardzo kosztowne, gdyż wymagać będzie nie tylko przeprojektowania i ponownego zaprogramowania określonych fragmentów systemu, ale również powtórzenia procesu testowania. Z tego powodu, sposób opracowania i udokumentowania specyfikacji wymagań ma podstawowe znaczenie dla powodzenia całego projektu. Poprawnie opracowana specyfikacja powinna mieć następujące cechy:

Jednoznaczność. Zapis każdego wymagania powinien mieć tylko jedną interpretację. Jako minimum wymaga się unikania synonimów dla określenia tego samego pojęcia oraz dołączenia słownika nazw wieloznacznych. Zaleca się również szerokie korzystanie z narzędzi sformalizowanych, takich jak diagramy stanu, grafy przepływu danych, sieci Petriego, tablice automatów, równania i nierówności.

Kompletność. Specyfikacja powinna wymieniać wszystkie istotne wymagania, jakie muszą być spełnione, a także definiować odpowiedź systemu we wszystkich sytuacjach na dowolne wartości danych wejściowych, zarówno poprawnych, jak i niepoprawnych.

Weryfikowalność. Opisy wszystkich wymagań powinny być tak formułowane, aby możliwe było jednoznaczne rozstrzygnięcie, czy produkt finalny spełnia te wymagania. Należy unikać określeń nieprecyzyjnych w rodzaju *„zazwyczaj"* lub *„na ogół"*.

Spójność. Opisane w specyfikacji wymagania nie mogą być wzajemnie sprzeczne. Wszystkie opisy odnoszące się do tych samych akcji lub obiektów muszą używać tej samej terminologii.

Modyfikowalność. Łatwość modyfikacji specyfikacji zależy głównie od stylu i organizacji tego dokumentu. Powinien on mieć przejrzysty podział na rozdziały, podrozdziały i punkty, zawierać spis treści i indeks oraz jawne odnośniki, łączące wzajemnie powiązane punkty. Każde wymaganie powinno być opisane tylko w jednym miejscu.

Powiązanie. Każde wymaganie zamieszczone w specyfikacji powinno być uzasadnione przez wskazanie przyczyny, z której ono wynika. Może to być np. numer punktu dostarczonej przez użytkownika wstępnej specyfikacji, numer paragrafu standardu lub normy jakości, itp. W celu umożliwienia powiązania specyfikacji z innymi dokumentami projektowymi, wszystkie jej punkty powinny być numerowane.

Wzorcowy układ specyfikacji [7] obejmuje następujące rozdziały:

Wstęp. Zawiera informacje ułatwiające korzystanie z pozostałej części dokumentu, w tym: zakres specyfikacji, definicje i skróty nazw, listę dokumentów związanych oraz streszczenie.

Opis ogólny. Zawiera intuicyjny opis wymagań i ograniczeń charakteryzujących projektowane oprogramowanie. Kolejne punkty opisują: otoczenie oprogramowania, tzn. związki z innymi systemami, sprzęgi komunikacyjne i metody współpracy, funkcje oprogramowania, charakterystykę użytkowników oraz takie ograniczenia, jak wymagania prawne, rodzaj sprzętu i oprogramowania systemowego, sprzęgi z innymi systemami, protokoły komunikacyjne, itp.

Wymagania szczegółowe. Dokładny opis wszystkich wymagań, które musi spełniać końcowy produkt. Poszczególne wymagania powinny tworzyć odrębnie numerowane punkty obejmujące:

- wymagania funkcjonalne, opisujące sposób przekształcania danych wejściowych w wyjściowe; dotyczy to opisu formatu danych oraz sposobu przetwarzania danych, zarówno prawidłowych jak i nieprawidłowych;
- wymagania dotyczące wydajności przetwarzania, opisujące statyczne i dynamiczne parametry oprogramowania, np. liczbę końcówek, liczbę jednocześnie przetwarzanych zbiorów, dokładność i czas reakcji systemu;
- wymagania dotyczące sprzęgów zewnętrznych, określające sposób współpracy z użytkownikiem, komunikację oprogramowania ze sprzętem oraz sposób współpracy z innymi elementami oprogramowania;
- ograniczenia projektowe;
- cechy szczególne, takie jak: organizacja bazy danych, prawa dostępu i notowanie śladu zdarzeń, środki zabezpieczające przed utratą danych, wymagania dotyczące pracy w warunkach częściowej awarii, możliwości restartu po awarii, spodziewane kierunki modyfikacji itp.

Indeks. Rozdział zawiera listę ważniejszych nazw i terminów, ze wskazaniem wszystkich punktów dokumentu, w których zostały użyte.

Opis projektu (ang. *software design description*) jest podstawowym dokumentem rozpoczynającym implementację programów. Jego zawartość powinna dokładnie określać sposób działania wszystkich jego części składowych (jednostek projektowych). Wymaga to określenia struktury oprogramowania, tzn. jego podział na jednostki projektowe, funkcje poszczególnych jednostek, sposób ich współpracy oraz sposób korzystania ze sprzęgów zewnętrznych.

Wzorcowy układ opisu projektu [8] obejmuje rozdział wstępny o treści zbliżonej do rozdziału wstępnego specyfikacji wymagań

oraz ciąg rozdziałów opisujących kolejno wszystkie jednostki projektowe. Opis każdej jednostki powinien być zbudowany według tego samego schematu oraz określać zdefiniowane w treści standardu atrybuty jednostki.

Zarówno definicja jednostki projektowej, jak i znaczenie atrybutów, są podane w treści standardu [8] dość ogólnie i mało konkretnie. Standard ten niezbyt przystaje też do rzeczywistości języka C, w którym obecnie powstaje większość oprogramowania sterującego. Dlatego, w dalszej części artykułu zostanie podany schemat dokumentacji, wzorowany wprawdzie na treści cytowanego standardu, lecz stosowany do specyfiki języka C. Przyjęto, że projekt dotyczy oprogramowania systemu rozproszonego, wyposażonego w wielozadaniowy system operacyjny czasu rzeczywistego, np. QNX lub OS-9.

Za jednostkę projektową będzie uważany każdy fragment oprogramowania, dla którego w treści projektu odrębnie określono wykonywane funkcje i metody testowania. Można wyróżnić następujące rodzaje tych jednostek:

- System – stanowi całość oprogramowania realizującego funkcje opisane w specyfikacji wymagań. Wynikiem projektu jest podział systemu na komponenty (podsystemy), o różnej strukturze wewnętrznej. Podział ten powstaje przez pogrupowanie funkcji określonych w specyfikacji wymagań i nie musi odpowiadać fizycznej strukturze sieci komputerowej.
- Komponent – stanowi fragment systemu realizujący grupę powiązanych funkcji, dla którego określono wymagania, metody współpracy z innymi komponentami i sposoby testowania. Komponentem może być np. grupa zadań, baz danych, zbiorów danych i obszarów wspólnych, tworzących podsystem rejestracji i archiwizacji danych pomiarowych lub grupa funkcji, zbiorów i struktur danych, tworzących bazę danych zbiorów indeksowych albo parser wchodzący w skład kompilatora. Komponenty dzielą się na komponenty prostsze, aż do poziomu zadań, zbiorów programowych, zbiorów danych i obszarów wspólnych.
- Zadanie – stanowi program sekwencyjny o określonych funkcjach, warunkach uruchomienia (aktywizacji) oraz sposobie współpracy z innymi zadaniami, zbiorami danych i obszarami wspólnymi. W języku C, zadanie jest programem, który może dzielić się na zbiory programowe i nagłówkowe.
- Zbiór danych – jest zapisanym na dysku zestawem danych o określonej strukturze wewnętrznej.
- Obszar wspólny – jest rezydującym w pamięci operacyjnej zestawem danych o określonej strukturze wewnętrznej. Obszar wspólny stanowi element sprzęgu między zadaniami.
- Zbiór programowy – jest zbiorem zawierającym deklaracje zmiennych oraz programy funkcji języka C. Zbiór programowy jest jednostką widoczną na poziomie języka C przez reguły dostępności zmiennych; jest również wyodrębnionym elementem fizycznej struktury programu.
- Zbiór nagłówkowy – jest zbiorem zawierającym definicje struktur i typów danych oraz stałych symbolicznych. Zbiór nagłówkowy nie jest jednostką programową języka C, ale wyodrębnionym elementem fizycznej struktury programu, który wymaga odpowiedniej dokumentacji.
- Funkcja C – jest podprogramem, komunikującym się z programem wywołującym przez parametry i zmienne wspólne.

Opis projektu składa się z opisów poszczególnych jednostek projektowych. Opis każdej treści składa się zawsze z dziesięciu punktów. Jeżeli treści określonego punktu nie można w chwili opracowania opisu określić, należy to wyraźnie zaznaczyć, pisząc np. „nie dotyczy" lub „zostanie określone później". Wspomnianymi punktami opisu są:

Nazwa – różna od nazw wszystkich innych jednostek.

Typ jednostki – określenie rodzaju jednostki projektowej. Dla zbiorów programowych i nagłówkowych oraz zbiorów danych i obszarów wspólnych należy tu podać również pełną nazwę ścieżkową katalogu (zbioru) zawierającego daną jednostkę.

Funkcja – krótkie określenie funkcji wykonywanej przez daną jednostkę systemu, bez wnikania w algorytm jej działania.

Przeznaczenie – krótkie określenie roli jednostki w systemie. Należy tu podać odsyłacze do tych punktów specyfikacji wymagań, dla spełnienia których jednostka ta została utworzona lub listę innych jednostek, które korzystają z tej jednostki, np. listę zadań wywołujących opisywaną funkcję, lub listę zbiorów programowych, które włączają opisywany zbiór nagłówkowy.

Struktura – określenie struktury projektu i oprogramowania. Należy tu podać listę jednostek niższego poziomu, jakie wchodzą w skład jednostki opisywanej. Opisy zbiorów programowych i nagłówkowych powinny zawierać listę zdefiniowanych funkcji, struktur danych oraz stałych symbolicznych.

Zależności – określenie jednostek niezbędnych do wykonania danej jednostki projektowej. Zależnie od rodzaju tej jednostki, należy podać tu listę jednostek inicjujących zadania, zbiory i obszary danych wspólnych, listę funkcji wywoływanych przez opisywaną funkcję lub listę zbiorów nagłówkowych, włączanych przez opisywany zbiór.

Sprzęg – określenie sposobu korzystania z jednostki. Zależnie od rodzaju opisywanej jednostki, należy tu podać procedurę instalacji i obsługi systemu lub podsystemu, sposób utworzenia zadania oraz sposób jego komunikacji z innymi zadaniami przez zbiory danych, obszary wspólne, wiadomości, sygnały, sprzęgi sieciowe, itp., sposób wywołania (nagłówek) funkcji wraz z opisem listy parametrów i zmiennych zewnętrznych, a także format odczytu i zapisu danych zbioru.

Zasoby – opis elementów zewnętrznych w stosunku do projektu, wykorzystywanych przez daną jednostkę. W tym punkcie należy wyliczyć kilka różnych rodzajów wymagań dotyczących zasobów, takich jak:

- wymagania kompilacyjne: biblioteki podprogramów standardowych, np. graficznych lub matematycznych, jednostki zapożyczone z innych projektów, kompilatory, systemy operacyjne,
- wymagania wykonawcze: typ komputera, system operacyjny, pojemność pamięci operacyjnej i dyskowej, urządzenia zewnętrzne,
- zużycie zasobów systemu: zajętość pamięci, czas wykonania;

Dane – opis wewnętrznych struktur danych jednostki. Stopień szczegółowości opisu jest ściśle skorelowany z opisem algorytmu. Należy tu udokumentować wszystkie elementy danych, widoczne w opisie algorytmu. Szczególnie dokładnie należy opisać struktury dynamiczne: listowe i grafowe.

Algorytmy – opis metod (reguł) przetwarzania danych. Należy tu podać algorytm złożenia funkcji jednostki składającej się z funkcji jednostek składowych lub algorytm przetwarzania danych prostej funkcji C. Preferowane są quasi formalne metody opisu przez podanie listy zdarzeń, grafu przepływu danych, sieci Petriego lub sieci działań.

Kontrola procesu zmiany

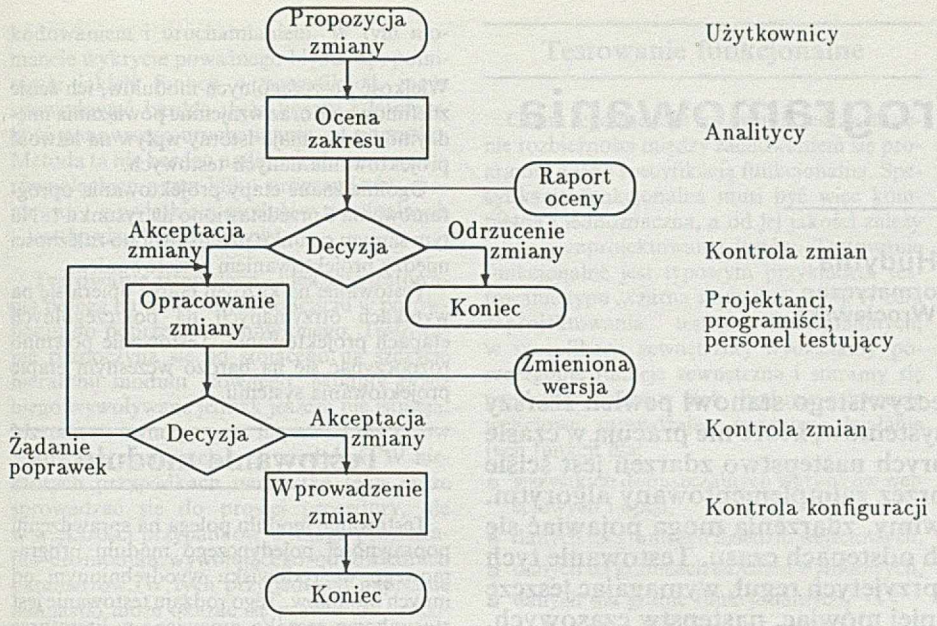
Opracowanie i akceptacja oprogramowania rozpoczyna okres eksploatacji oraz nieuchronnych zmian. Przyczyny zmian mogą być trojakie:

- błędy nie wykryte podczas testowania, a ujawnione podczas eksploatacji,
- zmiany wymagań użytkownika oraz potrzeba nowych lub innych funkcji,
- zmiany otoczenia, wywołane modyfikacją sterowanej instalacji.

Modyfikacja oprogramowania jest czynnością trudną i potencjalnie niebezpieczną. Każda zmiana narusza strukturę ustaloną podczas opracowania projektu oraz implementacji, a w dłuższym czasie prowadzi nieuchronnie do pogorszenia jakości oprogramowania i zmniejszenia jego podatności na dalsze zmiany. Z tego powodu procedura wprowadzania zmian powinna być starannie nadzorowana przez wyznaczoną do tego komórkę organizacyjną przedsiębiorstwa, odpowiedzialnego za funkcjonowanie oprogramowania. Żadna zmiana nie może być wprowadzona bez pisemnej akceptacji tej komórki. W przedsiębiorstwach eksploatujących instalacje potencjalnie niebezpieczne powinny dodatkowo istnieć specjalne komórki odpowiedzialne za kontrolę bezpieczeństwa instalacji, w tym kontrolę bezpieczeństwa zmian.

Procedurę wprowadzania i akceptacji zmian (rys. 1) inicjuje propozycja zmiany wysunięta najczęściej przez użytkowników, np. operatorów instalacji (którzy wykryli błędy) lub kierownictwo produkcji (które modernizuje instalację). Propozycja zmiany jest analizowana przez analityków systemu, którzy oceniają zakres wymaganych zmian, ich wykonalność i wymagane zasoby, a przede wszystkim czas oraz koszt tych zmian. Podstawę analizy stanowi opis projektu, zwłaszcza treści zawarte w punktach opisujących przeznaczenie i strukturę jednostek projektowych.

Wynikiem analizy jest raport oceny zmian, przedstawiany wraz z propozycją zmiany do zatwierdzenia komórce kontroli zmian. Każda zmiana, która narusza bieżące funkcjonowanie instalacji wymaga oczywiście akceptacji kierownictwa produkcji. Po akceptacji propozycji rozpoczyna się przygotowanie zmiany, które przebiega w sposób zbliżony do opracowania programu. W szczególności powinny być tu stosowane podobne metody i narzędzia wspomagające. Jeżeli zakres zmian jest duży, przygotowanie zmiany może być traktowane tak, jak nowe opracowanie, zawierające wszystkie fazy cyklu, przeglądy i dokumenty opisane w poprzednim rozdziale. Jeżeli zakres zmian jest mały i np. ogranicza się do skorygowania jednej tylko jednostki programowej, to wstępne fazy specyfikacji i projektu są zbędne.



Rys. 1. Procedura wprowadzania zmian do oprogramowania

Przygotowanie zmiany kończy się w każdym wypadku etapem testowania zmienionych jednostek i wszystkich zależnych od nich komponentów oraz całego systemu. Zakres testowania, którego koszt sięga połowy kosztu zmiany, wynika z opisu projektu, zwłaszcza z treści punktu opisującego zależności jednostek projektowych. Zmodyfikowana wersja programów jest przedstawiana komórce kontroli zmian do ostatecznej oceny i akceptacji. Dopiero po uzyskaniu takiej akceptacji, następuje wprowadzenie wszystkich zmian do znajdującego się w eksploatacji oprogramowania.

Komórkę kontroli zmian stanowi osoba lub zespół odpowiedzialny za eksploatację oprogramowania. Ponieważ zmiany – zwłaszcza duże – są kosztowne i mogą istotnie wpływać na bezpieczeństwo i efektywność pracy przedsiębiorstwa, komórka kontroli zmian powinna być umocowana w hierarchii organizacyjnej zakładu co najmniej na poziomie kierownictwa produkcji. W przypadku małych zmian, związanych z korygowaniem błędów i nie pociągających za sobą zmian funkcjonalnych, uprawnienia komórki mogą być przenoszone na niższe szczeble zarządzania, np. wprost na programistów. Zawsze jednak powinna istnieć jasno określona, osobista odpowiedzialność za wprowadzenie zmian.

Procedura wprowadzania zmian powinna być formalnie określona regulaminami służbowymi i bezwzględnie przestrzegana. Wszystkie dokumenty, takie jak: propozycja zmian, raport oceny zakresu, decyzje komórki kontroli zmian i dokumentacja zmienionej wersji, powinny mieć formę oficjalnych dokumentów pisemnych, o ustalonym formacie, podpisanych i zatwierdzonych przez ustalone osoby.

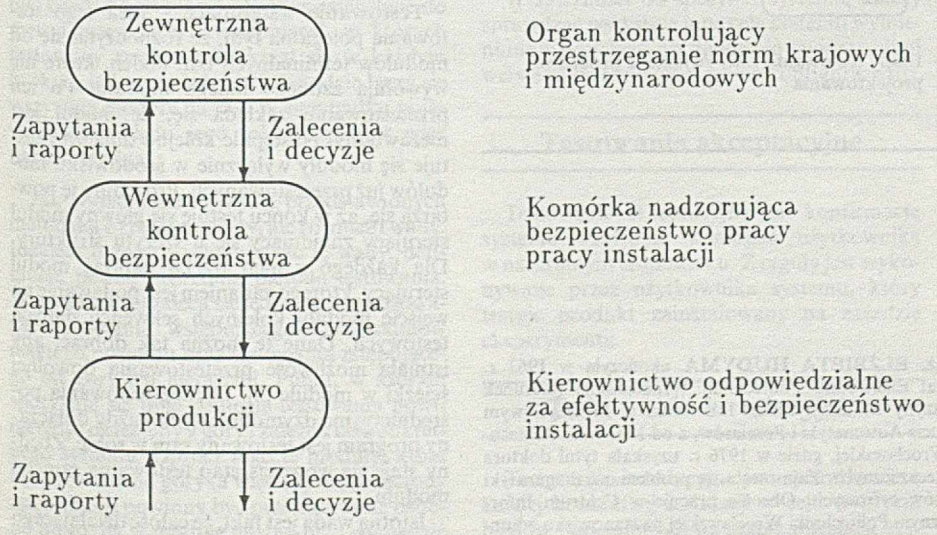
Każda wprowadzona do oprogramowania zmiana prowadzi do powstania nowej jego wersji. Nadzór nad kompletnością i spójnością różnych wersji należy do zadań komórki kontroli konfiguracji, która gromadzi zarówno moduły programowe, jak i związaną z nimi dokumentację.

Naszkicowany schemat kontroli zmian opisuje procedury wdrażane w przedsiębiorstwach zajmujących się opracowaniem oprogramowania. W przedsiębiorstwie eksploatującym wyłącznie oprogramowanie dostarczone przez zewnętrznego dostawcę, decyzja o wprowadzeniu zmiany ma charakter ekonomiczny i jest podejmowana przez kierownictwo produkcji.

Konserwacja oprogramowania sterującego instalacjami potencjalnie niebezpiecznymi, np. w fabrykach chemicznych, systemach kolejowych, lotniczych, itp. wymaga dodatkowej kontroli bezpieczeństwa instalacji, w tym kontroli bezpieczeństwa zmian. Komitet EWICS TC7 zaleca w tym zakresie trójpoziomowy system kontrolny, angażujący zarówno komórki wewnętrzne przedsiębiorstwa, jak i organizacje zewnętrzne (rys. 2).

- Kierownictwo produkcji zakładu lub wydziału eksploatuje instalację i odpowiada za jej bezpieczeństwo. Pdodejmuje decyzję o wprowadzeniu zmiany przygotowanej przez komórkę kontroli zmian, po zasięgnięciu opinii wewnętrznej kontroli bezpieczeństwa.
- Wewnętrzna kontrola bezpieczeństwa, umocowana na poziomie kierownictwa produkcji, analizuje wpływ zmiany na bezpieczeństwo pracy instalacji. Stosuje się do norm krajowych i międzynarodowych, a w sprawach nie uregulowanych normami zwraca się do zewnętrznej kontroli bezpieczeństwa.
- Zewnętrzna kontrola bezpieczeństwa jest resortowym lub krajowym ciałem normatywnym, określającym regulacje prawne, dotyczące np. norm poziomu emisji w fabrykach, wymaganych zabezpieczeń systemów kolejowych i lotniczych, itp.

Cała ta struktura decyzyjna może być zaangażowana w podejmowaniu decyzji zaznaczonych rombami na rys. 1. Decyzje konserwacyjne dotyczące przeglądów okresowych (przegląd sprzętu, testowanie spójności baz danych, itp.) oraz usuwania błędów oprogramowania w okresach zatrzymania pracy instalacji mogą być podejmowane samodzielnie przez komórkę kontroli zmian. Wszelkie decyzje dotyczące modyfikacji oprogramowania wymagają pisemnej akceptacji wewnętrznej kontroli bezpieczeństwa. Wątpliwości dotyczące zgodności z normami krajowymi i międzynarodowymi rozstrzygają organy zewnętrznej kontroli bezpieczeństwa.



Rys. 2. Struktura organów kontroli bezpieczeństwa

LITERATURA

[1] Sacha K.: Projektowanie oprogramowania systemów wbudowanych. INFORMATYKA, nr 7, 1993

[2] Smith I.: The integrity of computer based safety systems. Proc. IFIP/IFAC Conference on Hardware and Software for Real Time Process Control, Warszawa, 1988

[3] ANSI/IEEE Std 730-1989 Software Quality Assurance Plans

[4] ANSI/IEEE Std 983-1986 Guide for Software Quality Assurance Plans

[5] ANSI/IEEE Std 1028-1988 Standard for Software Reviews and Audits

[6] ANSI/IEEE Std 1058.1-1987 Standard for Software Management Plans

[7] ANSI/IEEE Std 830-1984 Guide for Software Requirements Specification

[8] ANSI/IEEE Std 1016-1987 Recommended Practice for Software Design Descriptions

[9] ANSI/IEEE Std 1063-1987 Standard for Software User Documentation

[10] ANSI/IEEE Std 828-1990 Standard for Software Configuration Management Plans

[11] ANSI/IEEE Std 1042-1987 Guide to Software Configuration Management

Testowanie oprogramowania

Elżbieta Hudyma
Centrum Informatyczne
Politechnika Wrocławska

Testowanie systemów czasu rzeczywistego stanowi pewien szerszy problem w stosunku do testowania systemów, które nie pracują w czasie rzeczywistym, tzn. takich, w których następstwo zdarzeń jest ściśle przewidywalne i określone przez zaimplementowany algorytm. W systemach, o których mówimy, zdarzenia mogą pojawiać się w dowolnej kolejności i w dowolnych odstępach czasu. Testowanie tych systemów przebiega według ogólnie przyjętych reguł, wymagając jeszcze dodatkowo przetestowania, najogólniej mówiąc, następstw czasowych. Sprowadza się to do konieczności wygenerowania na różnych etapach testowania dodatkowych danych testowych, zwiększając znacznie i tak duże koszty testowania.

Testowanie jest procesem wykonywania programu w celu wykrycia błędów. Za pomocą testowania nie można jednak udowodnić, że system poddany testom jest całkowicie wolny od błędów.

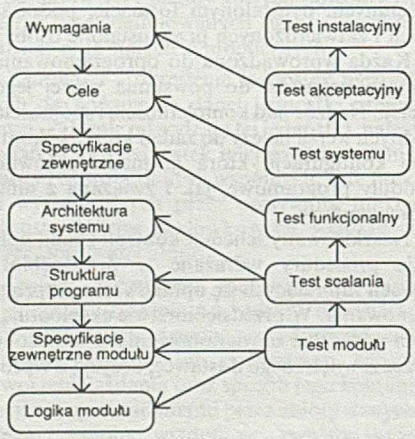
Proces testowania polega na wielokrotnym wykonaniu programu dla określonych zestawów danych wejściowych, zwanych **danymi testowymi.** Jednokrotne takie wykonanie programu nazywamy **przebiegiem testu** lub w skrócie – **testem.**

W ramach procesu testowania należy zaprojektować i zapisać dane testowe, wykonać program dla tych danych oraz porównać otrzymane wyniki przebiegu testu z przewidywanymi. Każdy test powinien być zapisany i przygotowany do ponownego użycia. W przeciwnym przypadku trudno jest przetestować program ponownie, np. po wprowadzeniu poprawek. Przed wykonaniem testu należy określić spodziewany wynik, a po jego wykonaniu – dokładnie porównać otrzymane wyniki ze spodziewanymi, zanim wynik testu zakwalifikuje się jako poprawny.

Konfirmacja jest próbą wykrycia błędów przez wykonanie programu w rzeczywistym środowisku.

Etapy testowania

Trudno przecenić wpływ jaki na testowanie mają projektowanie i struktura programu.



Etapy testowania i ich związek z procesami projektowania

Dr inż. ELŻBIETA HUDYMA ukończyła w 1967 r. Wydział Elektroniki Politechniki Wrocławskiej (kierunek – Maszyny Matematyczne). Pracowała w Przemysłowym Instytucie Automatyki i Pomiarów, a od 1973 r. na Politechnice Wrocławskiej, gdzie w 1976 r. uzyskała tytuł doktora nauk technicznych. Zajmowała się problemami diagnostyki systemów cyfrowych. Obecnie pracuje w Centrum Informatycznym Politechniki Wrocławskiej na stanowisku adiunkta.

Wielkość poszczególnych modułów, ich ścisłe zdefiniowanie oraz wzajemne powiązania międzymodułowe mają istotny wpływ na łatwość projektowania danych testowych.

Ogólnie znane etapy projektowania oprogramowania 5 przedstawiono na rysunku 1. Na tym samym rysunku przedstawiono zależności między projektowaniem a testowaniem.

Testowanie na każdym etapie opiera się na wynikach otrzymanych na poszczególnych etapach projektowania. Testowanie powinno rozpoczynać się na bardzo wczesnym etapie projektowania systemu.

Testowanie modułu

Testowanie modułu polega na sprawdzeniu poprawności pojedynczego modułu programowego w środowisku wyodrębnionym od innych modułów. Tego rodzaju testowanie jest stosunkowo szeroko omawiane w literaturze [2, 5, 6]. Chodzi o znalezienie rozbieżności między logiką i pośrednictwami modułu, a jego specyfikacją zewnętrzną. Przy tym testowaniu w dobrze zaprojektowanym systemie na ogół jednak nie spotyka się problemów typowych dla systemów czasu rzeczywistego, czyli problemów czasowych. Pozostaje problem znalezienia zbioru testów, który w dostateczny sposób sprawdzi poprawność działania modułu.

Testowanie scalania

Testowanie scalania jest weryfikacją pośrednictw między częściami systemu. Jedną z najistotniejszych kwestii przy scalaniu jest kolejność łączenia wszystkich modułów, ponieważ wpływa to na sposób tworzenia danych testowych, porządek kodowania modułów, rodzaj narzędzi stosowanych do testowania, a wreszcie na dokładność i koszt testowania. W przypadku testów scalania dla systemów czasu rzeczywistego pojawiają się typowe tam problemy następstw czasowych, co powinno być uwzględnione przy projektowaniu testów. Poniżej przedstawiono podstawowe strategie takiego działania wraz z krótkim omówieniem.

Testowanie wstępujące – idea tego testowania polega na tym, że rozpoczyna się od modułów terminalnych tzn. takich, które nie wywołują żadnego innego modułu. Po ich przetestowaniu zakłada się, że moduł jest niezawodny. Następnie kolejno dołącza i testuje się moduły wyłącznie w środowisku modułów już przetestowanych. Procedurę tę powtarza się, aż w końcu testuje się główny moduł sterujący znajdujący się u szczytu struktury. Dla każdego z nich trzeba napisać moduł sterujący, którego zadaniem jest podawanie na wejście modułu kolejnych sekwencji danych testowych. Dane te można tak dobrać, aby istniała możliwość przetestowania dowolnej ścieżki w module [2]. Oprócz testowania pośrednictw międzymodułowych każdy dołączany program jest testowany sam w sobie. Zbędny staje się wówczas etap testowania samego modułu.

Istotną wadą jest fakt, że całość działającego systemu osiąga się przy końcu pracy nad

kodowaniem i uruchamianiem. W tym momencie wykrycie poważnego błędu, np. pominięcie jakiejś funkcji w specyfikacji, może spowodować bardzo duże koszty i konieczność ponownego uruchamiania od początku. Metoda ta nie bardzo nadaje się do testowania równoległego przez kilku programistów, ponieważ – z wyjątkiem modułów terminalnych – jest bardzo hierarchiczna.

Testowanie zstępujące – idea tego testowania polega na odwrotnym podejściu w porównaniu do poprzednio omówionego. Testowanie rozpoczyna się od stojącego na szczycie hierarchii modułu głównego. Moduły przez niego wywoływane jednak jeszcze nie istnieją. Metoda ta wymaga więc napisania dla celów testowania namiastek tych modułów. W niektórych przypadkach namiastka taka może sprowadzać się do prostej procedury, ale w większości przypadków wymaga przekazania do modułu wywołującego odpowiednich parametrów. W tych przypadkach napisanie namiastki może okazać się prawie tak pracochłonne, jak symulowany moduł. Po przetestowaniu dołącza się do niego bezpośrednio wywoływane przez niego moduły i testuje się otrzymaną kombinację. Dane testowe są dostarczane w postaci danych użytkownika, pod warunkiem, że operacje wejścia-wyjścia będą dostępne na wczesnym etapie testowania. Testowanie to ma tę zaletę, że podstawowy szkielet programu i podstawowe funkcje specyfikacji zostają przetestowane na bardzo wczesnym etapie realizacji systemu, co pozwala na wczesne wykrycie większych błędów. Wadą tego typu testowania jest to, że wprowadzony do systemu moduł jest rzadko porządnie przetestowany, ponieważ wymagałoby to bardzo skomplikowanej namiastki, zwracającej wywołującemu ją modułowi odpowiednie sekwencje danych.

Testowanie równoczesne sprowadza się do osobnego przetestowania każdego modułu w izolacji od pozostałych modułów, a następnie scalenia ich wszystkich na raz. Metoda ta wymaga stosowania zatem zarówno danych testowych dla każdego modułu, jak i namiastek dla każdego wywoływanego modułu. Oprócz tego moduły scalane są na późnym etapie testowania powodując, że istotne błędy w pośrednictwach międzymodułowych pozostają długo niewykryte, a ich poprawienie może być bardzo kosztowne. W przeciwieństwie do metod zstępującej i wstępującej, w których testowany system powiększał się w każdym kroku o jeden moduł, w tej metodzie łączy się wszystkie moduły na raz. W przypadku pojawienia się błędu jego lokalizacja jest więc bardzo trudna.

Jak widać każda z wyżej przedstawionych metod ma określone zalety, ale również i wady. Ideałem byłoby wyeliminować te wady, definiując pewną pośrednią metodę testowania. W metodzie tej zaleca się jednoczesne stosowanie testowania wstępującego dla modułów niskiego poziomu oraz testowania zstępującego od góry ich hierarchii. Punkt, w którym spotykają się obie strategie testowania powinien być określony z góry i zależny od konkretnego systemu. Oprócz tego te moduły, które będą scalane od góry (a więc używając metody zstępującej) powinny być przetestowane osobno po to, by w module można było sprawdzić określone konstrukcje logiczne.

Testowanie funkcjonalne

Celem tego rodzaju testowania jest znalezienie rozbieżności między zachowaniem się programu, a jego specyfikacją funkcjonalną. Specyfikacja funkcjonalna musi być więc kompletna i jednoznaczna, a od jej jakości zależy łatwość zaprojektowania testów. Testowanie funkcjonalne jest typowym przykładem testowania typu „czarna skrzynka" [1, 5]. W celu zaprojektowania testów funkcjonalnych, w specyfikacji zewnętrznej wydzielamy poszczególne funkcje zewnętrzne i staramy się przebadać każdą z nich, budując sekwencje wejściowe, sprawdzające poprawne działanie tych funkcji dla:

- wszystkich dopuszczalnych warunków wejściowych i opcji,
- danych na granicach dziedzin wejściowych,
- danych na granicach dziedzin wynikowych,
- danych dla granic funkcjonalnych,
- danych niepoprawnych, niespodziewanych i destrukcyjnych.

Testowanie systemu

Testowanie systemu polega na znalezieniu rozbieżności między systemem, a jego celami początkowymi, najpierw w środowisku symulowanym (weryfikacja), a następnie w środowisku rzeczywistym (konfirmacja). Jest to możliwe tylko wtedy, gdy cele systemu były jasno wyspecyfikowane oraz mierzalne. Projektowanie danych testowych jest tu bardzo trudne, a podstawowego znaczenia nabiera sprawdzenie niektórych aspektów czasu rzeczywistego. Dla większości systemów należy przeprowadzić następujące testy:

- przeciążeń w każdym znaczeniu (np. ilościowym, czasowym itp.),
- objętościowe,
- ochrony danych,
- efektywności,
- niezawodności,
- funkcji użytych do podnoszenia systemu z załamania,
- dokumentacji.

W zależności od specyfiki systemu, należy sprawdzać wszystkie inne cele systemu wymienione w projekcie, np. zgodność z poprzednimi wersjami systemu, testowanie konfiguracji itp.

Testowanie akceptacyjne

Testowanie akceptacyjne jest konfirmacją systemu względem wymagań użytkownika w naturalnym środowisku. Z reguły jest wykonywane przez użytkownika systemu, który testuje produkt zainstalowany na zasadzie eksperymentu.

Testowanie instalacyjne

Testowanie instalacyjne jest sprawdzeniem każdej konkretnej instalacji w celu wykrycia błędów popełnionych na etapie instalacji systemu.

Projektowanie danych testowych

Największym problemem we wszystkich typach testów jest zaprojektowanie danych testowych. Na problem projektowania danych testowych można spojrzeć z dwóch różnych punktów widzenia:

- kryteriów wyboru danych testowych, które odnoszą się albo do funkcji, albo do struktury systemu, prowadząc bezpośrednio do testowania funkcjonalnego lub strukturalnego,
- sposobu generowania danych testowych, które mogą być deterministyczne lub probabilistyczne.

Testowanie funkcjonalne sprawdza program względem specyfikacji ignorując treść kodu. System jest traktowany jako „czarna skrzynka", tzn. że wewnętrzna struktura programu jest niedostępna [1, 5]. Tego rodzaju podejście do generowania testów można stosować na każdym etapie testowania. W tym przypadku należy zaprojektować testy, zarówno pozytywne, jak i negatywne. Te pierwsze polegają na sprawdzeniu działania programu przy sekwencjach testujących wybranych zgodnie ze specyfikacją, te drugie sprawdzają reakcję na nienormalne i niezgodne ze specyfikacją zdarzenia. Wadą tego rodzaju podejścia do generowania testów jest możliwość nieprzetestowania niektórych fragmentów kodu.

Przy testowaniu strukturalnym testy projektuje się biorąc pod uwagę strukturę programu, czyli system jest traktowany jako tzw. „szklana skrzynka" [2, 5]. Ten typ testowania nadaje się szczególnie na poziomie testowania modułów. Metoda opiera się na analizie dostępnych ścieżek w module. Następnie wyliczone predykaty ścieżki są porównywane ze specyfikacją, po czym następuje symboliczne wykonanie danego modułu. Zbiór testów powinien zapewniać wykonanie co najmniej raz każdej instrukcji. Ostatecznym celem jest przetestowanie każdej drogi w programie. Podstawową wadą metody jest to, że ignoruje ona zależności między modułami oraz nie wykrywa zmiany czasu wykonania logiki sterowania i brakujących ścieżek. Rozszerzenie tej metody na inne, niż testowanie modułów, etapy testowania jest bardzo trudne, ponieważ wymaga możliwości obserwacji wszystkich elementów systemu.

Kompletne przetestowanie systemu oznacza wykonanie go i sprawdzenie dla wszystkich możliwych kombinacji jego danych wejściowych. Nawet dla małego programu przy kilku parametrach wejściowych jest to praktycznie niewykonalne. W przypadku deterministycznego wyboru danych testowych, czynione są wysiłki przede wszystkim w celu zredukowania rozmiaru sekwencji testujących. Siłą rzeczy przy projektowaniu danych wejściowych należy wybrać ich określony podzbiór. Dobre dane testowe to takie, które z dużym prawdopodobieństwem wykryją nieznany dotąd błąd w programie, nie zaś takie, które pokażą, że program działa poprawnie. Dane testowe powinny obejmować zarówno dane poprawne jak i niepoprawne. Liczbę testów można zredukować opierając się na analizie systemu.

Przy projektowaniu sekwencji danych dla systemu czasu rzeczywistego należy brać pod uwagę wzajemne zależności masowe w sekwencji wejściowej, które są bardzo istotne ze

względu na synchronizację procesów. Prowadzi to do ogromnych rozmiarów sekwencji testowych. W testowaniu oprogramowania czasu rzeczywistego szczególne znaczenie ma [3]:

- pojawienie się zakleszczeń,
- poprawność sekwencji wyjściowej,
- pojawienie się końcowych lub pośrednich wyników w prawidłowym odcinku czasu.

Rozpatrzmy przypadek sekwencji zadań. Gdy M zadań ma dostęp do jednego zasobu, to istnieje $M!$ możliwych sekwencji dostępu. Jeśli przyjmiemy jednakowe prawdopodobieństwo błędu dla wszystkich dostępów, wtedy prawdopodobieństwo wykrycia jednego błędu dostępu w N przebiegach jest:

$$P = 1 - \left(1 - \frac{1}{M!}\right)^N.$$

Jeśli M zadań ma dostęp do n zasobów, to liczba możliwych sekwencji dostępu wynosi:

$$K = \sum_{j=1}^{n} \sum_{i=1}^{M} N_i \times m_{ij},$$

gdzie:

N_i – liczba przebiegów zadania i

m_{ij} – liczba dostępów do zasobu j podczas jednego przebiegu zadania i.

Jeśli chcemy rozważyć wszystkie możliwe dostępy, wtedy liczba tych sekwencji wynosi $K!$. Ten wynik określa liczbę testów, które należałoby wykonać, aby sprawdzić poprawność wszystkich możliwych dostępów.

Przy weryfikacji tych systemów czasu rzeczywistego, w których pojawiają się przerwania, występuje bardzo podobny problem. Pojawianie się w danym odcinku czasu przerwań o różnych priorytetach może inicjować sekwencję różnych zadań lub wymagać przeorganizowania kolejki. Takie przypadki są rozpatrywane kombinatorycznie w taki sam sposób, jak omówiony wyżej problem dostępu.

Opisany prosty model nie opisuje skomplikowanego zachowania się dużych systemów, a jak wskazano wyżej, wymaga bardzo dużej liczby testów dla uzyskania rozsądnego poziomu ufności. Testowanie systemu jest bardzo kosztowne dla dużych, rozproszonych systemów czasu rzeczywistego.

Drugim możliwym sposobem ograniczenia rozmiaru testów jest wybór określonej losowej próbki, czyli tzw. testowanie statystyczne [3, 6]. Podczas tego testowania program jest wykonywany N razy, przy czym dane wejściowe są generowane losowo zgodnie z określonym rozkładem prawdopodobieństwa. Liczba N nazywana jest tu rozmiarem testu.

Statystyczną jakością testu Q_N jest prawdopodobieństwo, że na N testów ujawni się co najmniej jeden błąd w programie. Q_N zależy od:

- istniejących błędów,
- wybranego rozkładu prawdopodobieństwa,
- rozmiaru testu N.

Niech p będzie średnim prawdopodobieństwem, że program załamie się na wybranej z rozkładu jednej z danych testowych. Wtedy dla wartości p prawdopodobieństwa załamania się programu, jakość testu Q_N oraz liczba testów N znajduje się w relacji:

$$(1 - p)^N = 1 - Q_n.$$

Stąd dla zadanego rozmiaru testu jakość wynosi:

$$Q_N = 1 - (1 - p)^N,$$

a poniższy wzór podaje minimalny jego rozmiar, wymagany do osiągnięcia wymaganej jakości testu:

$$N = \log(1 - Q_N)/\log(1 - p).$$

Istotnym problemem w powyższych rozważaniach jest oszacowanie parametru p. Niestety, przy obecnym stanie wiedzy, nie wydaje się możliwe zdefiniowanie poprawnego modelu niezawodności oprogramowania tak, aby można było a priori oszacować prawdopodobieństwo załamania systemu na podstawie wykrytych błędów. Stosuje się różne metody zastępcze, np. określenie p na podstawie częstotliwości użycia każdej instrukcji, przez wysianie sztucznych błędów w programie itp.

Przyjmijmy za [6] następujące wartości: $p = 10^{-5}$, a $Q_N = 0{,}999$. Wtedy wymaganą liczbę testów można obliczyć podstawiając do wzoru: $N = \log 0.001/\log 0.99999$, co wynosi około 690 000 testów.

Im mniejszy rozmiar próbki, tym mniejsze prawdopodobieństwo wykrycia błędów w programie. Redukując koszty testowania przez statystyczny dobór sekwencji testujących zwiększamy ryzyko pozostania błędu w systemie dla przypadków, które nie są sprawdzane przez test.

Testowanie dużych systemów ma swoje dodatkowe wymagania w porównaniu do testowania systemów prostych. Niektóre z tych wymagań są następujące:

- Testowanie własności czasu rzeczywistego zależy od implementacji (oraz sprzętu). Dlatego system trzeba testować jako całość.
- Analiza formalna jest podstawową metodą pozwalającą na redukcję sekwencji testujących dla systemów, którym należy zapewnić wysoki poziom ufności.

Narzędzia wspomagające testowanie

Podstawowe problemy związane z testowaniem, to przede wszystkim ogromna liczba testów, które należy zaprojektować, zapamiętać, wykonać i ocenić wynik. Powoduje to, że koszty testowania są tak wysokie. Aby przyspieszyć i usprawnić testowanie, wykorzystuje się często komputerowe narzędzia testujące, które uaktywniają testowany system. Niestety rzadko narzędzia te są uniwersalne, a koszt budowy specjalnych narzędzi i systemów testujących często dorównuje, a nawet przewyższa koszty budowy samego systemu. Urządzenie testujące generuje wejściowe sekwencje testowe, steruje odmierzaniem odcinków czasu (ang. *timer*) oraz monitoruje wyjścia. Dlatego też niezawodność urządzenia testującego powinna być bardzo wysoka. Jeśli urządzenie testujące wymaga generatora liczb losowych, to zakres generowanych liczb musi być odpowiednio duży.

Możliwość wprowadzania punktów kontrolnych w systemach czasu rzeczywistego jest znacznie ograniczona. W celu śledzenia przepływu sterowania lub przepływu danych należy wstawić dodatkowe instrukcje do programu, wprowadzając w ten sposób zmiany w środowisku działającym w trybie czasu rzeczywistego. W systemach o ostrym reżimie czasu rzeczywistego tego rodzaju metody wspomagające testowanie są niemożliwe do zastosowania.

W systemach czasu rzeczywistego testowanie często nie może na żadnym etapie odbywać się w warunkach naturalnych. W takich przypadkach testowanie powinno odbywać się w środowisku symulowanym [3]. Budowa takich symulatorów „życia rzeczywistego" jest oczywiście bardzo kosztowna.

Można wyróżnić dwie klasy symulacji używane do testowania systemów czasu rzeczywistego:

- symulacja testowanego systemu,
- symulacja środowiska lub obciążenia, w którym system ma pracować.

Symulacja testowanego systemu pozwala na jego przebadanie we wczesnym etapie projektowania, przewidzenia zachowania się tego systemu na pewnym poziomie ufności, porównanie po realizacji projektu osiągniętych wyników z projektowanymi itp. Podstawowym problemem jest koszt i wysiłek tego rodzaju symulacji, który w wielu wypadkach dorównuje lub przewyższa koszt samego systemu.

Częściej spotykany jest symulator środowiska, który jest używany przy pomiarach systemu dla określonych obciążeń, a przy testowaniu systemu wtedy, gdy jest ono w rzeczywistym środowisku zbyt ryzykowne (np. w elektrowniach jądrowych) lub dla odtworzenia warunków prowadzących do zakleszczeń lub załamań systemu.

Słabe punkty symulacji, to możliwość popełnienia tych samych błędów, zarówno przy projektowaniu systemu, jak i symulatora tego systemu, złożoność dużych symulatorów, konieczność weryfikowania symulatora jako narzędzia weryfikacji, trudność rozróżnienia między błędnym systemem, a błędnym symulatorem.

Symulator środowiska jest nieocenionym narzędziem przy testach funkcjonalnych oraz testach systemu.

* * *

Przedstawiono metody testowania systemów czasu rzeczywistego jako sprawdzanie zgodności między specyfikacją poszczególnych etapów projektowania a implementacją. Możliwości, a przede wszystkim koszty testowania zależą w dużym stopniu od określonych własności takich programów, co jest szczególnie istotne w systemach wieloaplikacyjnych, adaptacyjnych i tolerujących uszkodzenia. Pożądane cechy systemu czasu rzeczywistego są wymienione poniżej.

- System może zawierać pewną liczbę aplikacji niezależnie działających.
- Systemy operacyjne czasu rzeczywistego powinny umożliwiać weryfikację każdej z aplikacji niezależnie.
- Dodanie niezależnego modułu do systemu nie wymaga testowania wszystkich aplikacji.
- Relacja między systemem operacyjnym a aplikacjami powinna być spójna, co sprawia, że założenia w obu dziedzinach nie są sprzeczne.
- Dopuszczona przez system do wykonania aplikacja ma zagwarantowane, że zostanie wykonana poprawnie, niezależnie od innych aplikacji, dzielących z nią zasoby systemu.

Jak już powiedziano, testowanie jest procesem bardzo kosztownym, a jednym z najtrudniejszych jego problemów jest decyzja, kiedy je zakończyć; dotyczy to każdego etapu testowania. Ponieważ kompletne przetestowanie systemu nie jest możliwe, należy wyznaczyć jasne kryteria jego zakończenia. Decyzję o zakoń-

Wykaz referatów opublikowanych w materiałach konferencyjnych

I. Bezpieczeństwo i niezawodność oprogramowania

prof. dr hab. inż. Janusz GÓRSKI	„Bezpieczeństwo oprogramowania – niektóre aktualne problemy badawcze"
mgr inż. Zdzisław ŻURAKOWSKI	„Systemy komputerowe w zastosowaniach związanych z bezpieczeństwem"
dr hab. inż. Eugeniusz KURIATA	„Systemy odpowiedzialne za bezpieczeństwo"
dr inż. Ireneusz JÓŹWIAK	„Niezawodność oprogramowania"
dr inż. Daniel ARENDT	„Obsługa zdarzeń wyjątkowych i rejestracja błędów w programie czasu rzeczywistego"

II. Analiza i specyfikacja wymagań

dr inż. Krzysztof SACHA	„Specyfikacja wymagań systemu czasu rzeczywistego metodą Warda-Mellora"
prof. dr hab. inż. Tomasz SZMUC	„Graficzna specyfikacja oprogramowania czasu rzeczywistego"
mgr inż. Piotr SZWED	
mgr inż. Stefan ROZMUS	„Metoda komputerowego wspomagania w ustalaniu kryterium zdatności użytkowej stałoprogramowego systemu komputerowego"
mgr inż. Grzegorz BLIŹNIUK	„Model formalny systemów czasu rzeczywistego w postaci VDM"

III. Metody projektowania

dr inż. Jan KWIATKOWSKI Damian MISIAK	„Metoda strukturalnej analizy i projektowania systemów z uwzględnieniem rozszerzeń dla systemów czasu rzeczywistego"
dr inż. Zbigniew ZIELIŃSKI	„Technologia budowy pokładowych lotniczych systemów sterowania"
mgr inż. Dariusz GAWRYCH	„Pakiet komunikacji programu z otoczeniem. Wykorzystanie mechanizmu skrzynki systemu VMS do wymiany danych w systemach czasu rzeczywistego"

IV. Języki, systemy operacyjne, narzędzia wspomagające

dr inż. Elżbieta KOSMULSKA-BOCHENEK	„Synchronizacja w rozproszonych systemach operacyjnych czasu rzeczywistego"
dr inż. Zbigniew FRYŹLEWICZ dr inż. Jędrzej UŁASIEWICZ	„Języki Ada, Modula-2 i system QNX jako narzędzia tworzenia aplikacji czasu rzeczywistego"
mgr inż. Wiesław BARCIKOWSKI	„System operacyjny QNX w rozproszonych systemach czasu rzeczywistego"
mgr inż. Piotr S. MICHAŁOWSKI	„Zastosowanie systemu operacyjnego QNX w wersji 4.2. do tworzenia rozproszonych symulatorów szkoleniowych"
mgr inż. Wojciech KULAS	„Środowisko inżynierii programowania COHESION w praktyce"

V. Normalizacja

dr inż. Lesław SIENIAWSKI	„Norma IEC: Oprogramowanie przemysłowych systemów związanych z bezpieczeństwem"

VI. Sprzężenie człowiek-system komputerowy: Ergonomia – wytyczne i przykłady rozwiązań

dr inż. Andrzej KALIŚ mgr inż. Alina TOPCZEWSKA	„Projektowanie styku z użytkownikiem w systemach czasu rzeczywistego"
dr inż. Marek SALOMON Cezary KOWALSKI	„Pokładowy system zobrazowania"

VII. Opis systemów projektowanych lub zrealizowanych

mgr inż. Piotr E. BRUDŁO prof. dr hab. inż. Henryk KRAWCZYK dr inż. Piotr SZPRYNGIER mgr inż. Piotr W. UMIŃSKI	„Projektowanie i wdrożenie mikroprocesorowego systemu współbieżnego na przykładzie urządzenia Systemu Selektywnego Wywołania"
mgr inż. Mariusz POSTÓŁ	„Struktura przemysłowych sieci komputerowych a ich niezawodność na przykładzie rozproszonego systemu monitorowania Elektrociepłowni IV w Łodzi"
dr inż. Jacek SZYDŁOWSKI mgr inż. Sławomir JASIŃSKI mgr inż. Mariusz MACIEJEWSKI	„System zdalnego sterowania i kontroli dyspozytorskiej metra warszawskiego"
dr inż. Jan GERTZ dr inż. Maciej PUCHAŁA	„Oprogramowanie dla podsystemu zobrazowania danych w komputerowych systemach kontroli dyspozytorskiej"
mgr inż. Henryk A. KOWALSKI dr Andrzej SKORUPSKI	„Wieloprocesorowy system przetwarzania sygnałów analogowych"
mgr inż. Paweł GAJKOWSKI mgr inż. Wojciech WAWER mgr inż. Andrzej DYLĄG	„Oprogramowanie systemu sterowania piecami do wypalania porcelany na bazie ekspertowego systemu operacyjnego czasu rzeczywistego RTES"
mgr inż. Stanisław KACZMAREK mgr inż. Leszek NASKRĘT	„Mikrokomputerowy system automatyki PRO-2000"

czeniu testowania można podejmować na podstawie:

- oszacowania pozostałych błędów w systemie,
- upływu arbitralnie ustalonego terminu zakończenia testowania,
- wykrycia w systemie określonej liczby błędów,
- rodzaju rozkładu wykrywalnych błędów w czasie testowania.

LITERATURA

[1] Ghiassi M.: Functional Testing of Optimizing C Compilers. Proceedings of the IFIP WG 5.4 Working Conference on Approving Software Products, North-Holland 1990

[2] Kuchta S.: Systematyczne testowanie programów. Informatyka nr 4 i 5, 1991

[3] Levi S.T., Agrawala A.K.: Real Time System Design. Mcgraw-Hill Publishing Company 1990

[4] Milovanovic R., Kulenovic A. and others: Approving Real Time Software – Current Practice and Trends. Proceedings of the IFIP WG 5.4. Working Conference on Approving Software Products, North-Holland 1990

[5] Myers G.J.: Projektowanie niezawodnego oprogramowania. WNT, Warszawa 1980

[6] Thevenod-Fosse P.: On the Efficiency of Statistical Testing with respect to Software Structural Test Criteria. Proceedings of the IFIP WG 5.4. Working Conference on Approving Software Products, North-Holland 1990

Przypominamy!

Wszystkie numery archiwalne Informatyki można nabyć w siedzibie redakcji

Metody zbiorów rozmytych w analizie decyzyjnej przedsiębiorstwa Część 1

Tadeusz Witkowski
Kombinat Przemysłu Narzędziowego VIS
Warszawa

Coraz szersze zastosowanie techniki komputerowej do rozwiązywania problemów podejmowania decyzji stwarza również potrzebę opracowania nowych podejść do problemu przetwarzania niepełnej informacji. Większa część tego typu informacji jest niedostępna w formie dokładnej tzn. w postaci ściśle określonych liczb, a jedynym jej źródłem w większości przypadków jest ekspert. Dlatego też jednym z istotnych zadań współczesnej informatyki jest obecnie opracowanie sposobów i metod reprezentacji i pozyskiwania wiedzy w warunkach niepełnej i niedokładnej informacji.

Teoria podejmowania decyzji rozwija się w ostatnich latach w kierunku tzw. nieostrego modelowania [6, 8, 10, 15]. Stwarza to możliwość utrzymania większej stabilności modeli w granicach nieostrych parametrów oraz przejścia, drogą adaptacji nowych informacji, do modeli coraz bardziej ostrych. Jest to bardzo istotne ze względu na to, że w zastosowaniu komputeryzacji do procesów decyzyjnych należy usunąć bariery między „nieostrym", „miękkim" sposobem myślenia człowieka, a „ostrymi" decyzjami komputerów.

Najczęściej spotykanymi metodami podejmowania decyzji przy niepełnej informacji są:

- metoda zbiorów rozmytych (ang. *fuzzy sets theory*),
- metoda niepełnej informacji liniowej – NIL (*ang. LPI linear partial information*). Metody te należą do dziedziny nieostrego modelowania.

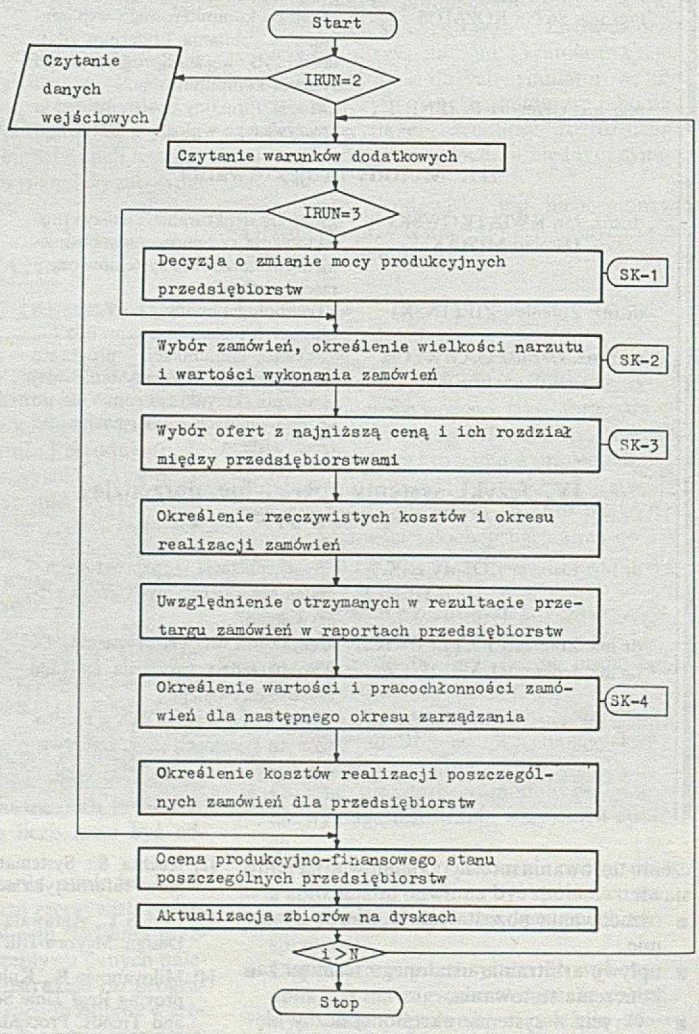
Szczególną rolę w reprezentacji i pozyskiwaniu wiedzy przy podejmowaniu decyzji mają metody i modele zbiorów rozmytych [2, 3, 4, 7, 10, 11, 12, 13, 14, 15]. Większość modeli podejmowania decyzji w warunkach rozmytych ma charakter normatywny i dotyczy formalizacji etapu wyboru, kiedy zbiór alternatyw, kryteria ocen, ograniczenia, relacje uporządkowania i inne, są określone. Zgodnie z klasyfikacją [6, 10], istniejące podstawowe modele podejmowania decyzji w otoczeniu rozmytym można podzielić na następujące niezależne grupy:

- indywidualne i zespołowe (na podstawie liczby decydentów),
- jednokryterialne i wielokryterialne (na podstawie liczby wykorzystywanych kryteriów),
- jednoetapowe i wieloetapowe (na podstawie liczby etapów podejmowania decyzji),
- rozmytego programowania matematycznego i rozmytych relacji boolowskich (na podstawie charakteru opisu priorytetów),
- lingwistyczne (wykorzystujące logikę rozmytą z lingwistycznymi wartościami prawdy),

W artykule zostaną rozpatrzone niektóre rozmyte modele, wykorzystywane na etapie pozyskiwania i reprezentacji wiedzy w procesie podejmowania decyzji, na przykładzie problemu modelowania działalności finansowo-produkcyjnej przedsiębiorstwa, uzyskującego zamówienia w formie przetargu.

Sformułowanie problemu analizy decyzyjnej

Rozpatrzmy problem zgodnie z warunkami gry decyzyjnej przedstawionej w [17, 21, 22] oraz na rys. 1.



Rys. 1. Schemat blokowy gry decyzyjnej modelującej działalność finansowo-produkcyjną przedsiębiorstw w przetargu,
IRUN-2 – inicjacja gry, IRUN-3 – podejmowanie decyzji o zmianie mocy produkcyjnej, N – liczba okresów gry; SK1–SK4 – monitory

Istnieje zbiór zamówień $Z = \{Z_i\}$, gdzie $i \in 1, n$, które przedstawiono do przetargu przedsiębiorstwom rywalizującym o ich otrzymanie. Każde przedsiębiorstwo ze zbioru $K = \{K_k\}$, gdzie $K \in 1, m$, w zależ-

ności od zdolności produkcyjnych własnych oraz podwykonawców, przedstawia ofertę na realizację określonej liczby zamówień oraz określa je spośród pełnego zbioru ofert. Każde przedstawione przez organizatora i-te zamówienie charakteryzuje się rozmytą wartością jego realizacji $\tilde{T}_i$ oraz rozmytym czasokresem wykonania $\tilde{W}_i$.

W każdym okresie podejmowania decyzji czasokres oraz wartość realizacji każdego zamówienia są generowane za pomocą generatora liczb losowych, odpowiednio o równomiernym i normalnym rozkładzie, w określonych dość ściśle przedziałach. Liczba zamówień w każdym okresie podejmowania decyzji może być zarówno stała, jak i zmienna, a ich generowanie przy podstawowym wariancie gry decyzyjnej jest określane przez stosunek $N_1{:}N_2{:}N_3{:}N_4$ (np. 4, 3, 2, 1), przy czym największą liczbę zamówień stanowią prace małe, następnie średnie itp. Wszystkie przedstawione do przetargu zamówienia charakteryzują się odpowiednią strukturą i można je określić w następujący sposób:

Z_1 – zamówienia małe, $Z_1 = \{\tilde{T}_1, \tilde{W}_1\}$,
Z_2 – zamówienia średnie, $Z_2 = \{\tilde{T}_2, \tilde{W}_2\}$,
Z_3 – zamówienia duże, $Z_3 = \{\tilde{T}_3, \tilde{W}_3\}$,
Z_4 – zamówienia bardzo duże, $Z_4 = \{\tilde{T}_4, \tilde{W}_4\}$,

gdzie $\tilde{T}_i, \tilde{W}_i$ – rozmyte oznaczone tyldą czasokresy i wartości realizacji zamówienia określonego typu Z_i, $i \in 1,4$. Przyjmuje się, że $\tilde{T}_1 \leqslant \tilde{T}_2 \leqslant \tilde{T}_3 \leqslant \tilde{T}_4$ oraz $\tilde{W}_1 \leqslant \tilde{W}_2 \leqslant \tilde{W}_3 \leqslant \tilde{W}_4$, chociaż mogą być generowane również inne relacje, podobnie jak w przypadku określenia pełnego zbioru zamówień, w którym $N_1 \geqslant N_2 \geqslant N_3 \geqslant N_4$.

Podejmowane przez każde przedsiębiorstwo decyzje dotyczą następujących problemów:

- ile ofert przedstawiać w przetargu?
- jakiego rodzaju zamówienia wybrać?
- jaką cenę (określaną wysokością narzutu) należy przedstawić w każdym z zamówień?
- czy i w jakim procencie należy zwiększyć lub zmniejszyć moce produkcyjne własnego przedsiębiorstwa lub przekazać część zamówień innym wykonawcom?

Działalność przedsiębiorstw jest oceniana na podstawie kryteriów opisanych w dalszej części artykułu.

Na początku przetargu przedsiębiorstwa określają koszty i cenę realizacji zamówienia, zapewniające osiągnięcie zysku. Jeżeli przedsiębiorstwo bardzo dąży do wygrania przetargu, proponując ofertę z niskim narzutem, przy podobnych kosztach produkcji jak inne przedsiębiorstwa, to szansa wygrania jest bliska 100%. Gdy narzut jest bardzo wysoki, prawdopodobieństwo wygrania przetargu jest równe 0. Jest zrozumiałe, że między tymi dwoma ekstremalnymi sytuacjami istnieją takie, dla których prawdopodobieństwo wygrania przetargu może zawierać się w przedziale od 0 do 100%.

Modele rozmytego programowania matematycznego i rozmytej oczekiwanej użyteczności

Zwykle pod pojęciem programowania matematycznego rozumie się zadanie odszukania wartości ekstremalnej określonej funkcji celu na dopuszczalnym zbiorze alternatyw. Za pomocą funkcji celu formalnie przedstawia się jedną z podstawowych właściwości: użyteczność, jakość itp. Rozmytość w formułowaniu problemu programowania matematycznego może być zawarta w opisie zbioru alternatyw, jak i w opisie funkcji celu. Różnorodne formy przedstawiania informacji wyjściowej warunkują istnienie sformułowań zadań rozmytego programowania.

Zgodnie z [6, 10] zadanie osiągnięcia rozmytego celu przy rozmytym ograniczeniu jest rozwiązywane następująco. Rozmyty cel G i rozmyte ograniczenie C przedstawia się za pomocą rozmytych podzbiorów uniwersalnego zbioru alternatyw X, tj. odpowiednio funkcjami $\mu_G: X \to [0,1]$ i $\mu_C: X \to [0,1]$, przy czym $F(X) = \{\mu | \mu: X \to [0,1]\}$. Rozwiązanie rozmyte określa się przy tym jako rozmyty podzbiór D zbioru X, otrzymanego w wyniku połączenia rozmytych celów i rozmytych ograniczeń decydenta, $\mu_D = \mu_G * \mu_C$, gdzie $*$ to określona operacja boolowska na F(X).

W problemach zarządzania rozmyte programowanie najczęściej wykorzystuje się do określania struktury programu produkcyjnego przedsiębiorstwa [6, 11, 20]. Rozwiązanie tego problemu polega na znalezieniu najlepszego rozwiązania kompromisowego, przy czym niedokładność takich danych, jak ceny, wielkości zbytu produkcji, pracochłonność technologii i zużycie materiałów, wiąże się z koniecznością wprowadzenia współczynników rozmytych ze względu na niepewność, zarówno w ograniczeniach, jak i w funkcjach kryterialnych.

Tak więc należy rozwiązać problem wielokryterialnego programowania liniowego z rozmytymi współczynnikami, który ogólnie można zapisać w następujący sposób:

$$\text{MAX} \left| \begin{matrix} \tilde{c}_1^T x \\ \ldots. \\ \tilde{c}_k^T x \end{matrix} \right|$$

przy ograniczeniach $\tilde{a}_i x - \tilde{b}_i$, $i = 1, \ldots, m'$; $\tilde{a}_i x - \tilde{b}_i$, $i = m'+1, \ldots, m$, gdzie: x – zmienne decyzyjne; $\tilde{c}_1, \ldots, \tilde{c}_k$ – współczynniki kosztu funkcji celu; $\tilde{a}_i$, $i = 1, \ldots, m$ – wiersze macierzy ograniczeń; $\tilde{b}_i$, $i = 1, \ldots, m$ – prawe strony ograniczeń.

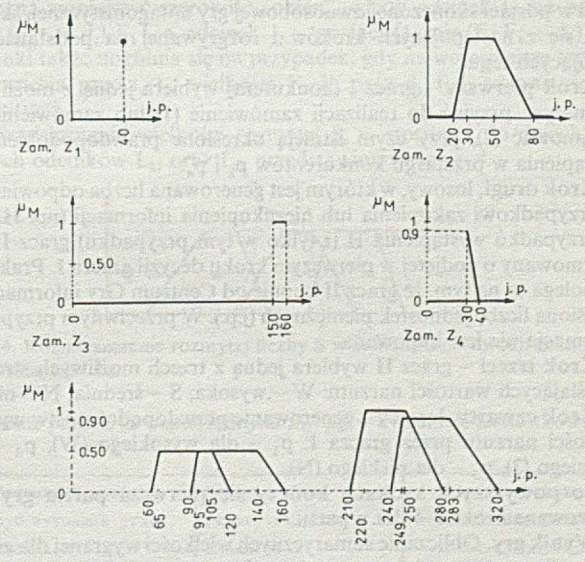
Przy opracowaniu przez przedsiębiorstwo strategii wyboru struktury zamówień dokładnie analizowane są różne warianty i możliwości otrzymania określonej wartości zysku w wyniku ich realizacji. W takim przypadku wygodnie jest zastosować do tego celu aparat teorii zbiorów rozmytych. Każda rozmyta wartość jest rozpatrywana tak, jak iloczyn zbiorów rozmytych o postaci trapezów. Każdy z takich rozmytych zbiorów M_i jest przedstawiony za pomocą uporządkowanej piątki $M_i = (m_i', m_i'', \alpha_i, \beta_i, h_i)$, gdzie: m_i' i m_i'' – odpowiednio dolne i górne wartości rozmytego zbioru M_i, α_i; β – lewy i prawy współczynnik rozmytości; h_i – wysokość zbioru rozmytego M_i.

Przykład 1

Zgodnie z tymi oznaczeniami i charakterystyką źródeł zysku, rozmyte wartości związane z nimi mogą być przedstawione w postaci: $M_i = (m_i', m_i'', \alpha_i, \beta_i, h_i)$, $M_j = (m_j', m_j'', \alpha_j, \beta_j, h_j)$ itd.

Zgodnie z [3], liczbą rozmytą $M_i \oplus M_j$, gdzie M_i i M_j to dwa zbiory rozmyte o postaci trapezu, jest zbiór rozmyty o postaci trapezu $(m', m'', \alpha, \beta, h)$, gdzie $h = \min(h_1, h_j)$ to efekt ścięcia, przy czym: $\alpha = h(\alpha_i/h_i + \alpha_j/h_j)$, $\beta = h(\beta_i/h_i + \beta_j/h_j)$, $m' = m_i' + m_j' - \alpha_i - \alpha_j + \alpha$, $m'' = m_i'' + m_j'' + \beta_i + \beta_j - \beta$, a iloczyn zbiorów rozmytych równa się $S = \bigcup_{i=1}^{i=n} M_i$ (gdzie n – liczba źródeł zysku).

Przedstawione w przetargu zamówienia mają w naszym przypadku (rys. 2) różne charakterystyki opisujące rozmyte wartości otrzymania zysku w wyniku ich realizacji.



Rys. 2. Rodzaje źródeł zysku różnorodne zamówienia opisane za pomocą rozmytych wartości oraz ich wartość sumaryczna

Wspólnie z modelami programowania matematycznego, do opisu indywidualnego procesu podejmowania decyzji, jest szeroko stosowana teoria decyzji statystycznych oraz teoria oczekiwanej użyteczności. Teoria oczekiwanej użyteczności jest stosowana w przypadkach, gdy nieokreśloność jest uwarunkowana brakiem obiektywnej skali dla oceny preferencji alternatyw.

Oprócz możliwości określenia rozmytej maksymalnej i minimalnej wartości zysku, przy wyborze struktury zamówień uwzględnia się także prawdopodobieństwa otrzymania każdego rodzaju zamówień. W tym celu w bazie wiedzy systemu eksperckiego [21, 22] znajdują się reguły

produkcji wykorzystujące teorię Bayesa do określenia prawdopodobieństwa $p(H_i, E_i)$ sytuacji, w której przedsiębiorstwo X (uczestnik przetargu) przedstawiło ofertę na zamówienie Z_i. W bazie wiedzy tego systemu znajdują się reguły o postaci: „IF na zamówienie Z_i złożono ofertę, THEN prawdopodobnie przedstawiło je przedsiębiorstwo X". W regułach tych przesłankami są pewne zdarzenia E_i, a wnioskami hipotezy – H_i. Przy określeniu prawdopodobieństwa $p(H_i, E_i)$ wykorzystuje się jednocześnie dane statystyczne charakteryzujące poprzednie etapy modelowania.

W wielu pracach niepełna informacja o funkcji rozkładu prawdopodobieństwa jest modelowana za pomocą liczb rozmytych i prawdopodobieństw lingwistycznych. W tym celu oprócz rozmytych ocen Bayesa wykorzystuje się loterie rozmyte, rozmyte drzewa preferencji, teorię możliwości oraz inne podejścia.

Rozmyte modele decyzji zespołowych

W zależności od rodzaju indywidualnych preferencji oraz aspektów informacyjnych wyróżnić można trzy klasy teorii podejmowania decyzji przez n decydentów:
1) teoria decyzji grup,
2) teoria małych grup,
3) teoria gier.

Zastosowanie teorii wyboru grupowego w warunkach rozmytości do procesów podejmowania decyzji, na przykładzie zakupu produktu w przetargu i wyboru struktury programu produkcyjnego, przedstawiono m.in. w [18].

Do modelowania procesów podejmowania decyzji w warunkach uwzględnienia interesów wszystkich decydentów mających wpływ na przebieg zdarzeń stosuje się aparat teorii gier.

Grę dwóch decydentów z rozmytymi strategiami i preferencjami określa się przez zestaw $G = (S^1, S^2, R^1, R^2)$, gdzie S^i – zbiór strategii i-tego gracza, $i \in 1,2$. Wyniki gry $\omega \in \Omega$ są określane na iloczynie kartezjańskim strategii graczy $\Omega = S^1 \times S^2$. Relacje i-tego gracza są określane za pomocą relacji rozmytej $\mu_{R^i}: \Omega \times \Omega \to [0,1]$.

Przykład 2

Rozpatrzmy model do określenia ceny realizacji zamówienia w przetargu w postaci skończonej dwuosobowej gry antagonistycznej, składającej się z następujących kroków i rozgrywanej na podstawie następujących reguł:

1. Krok pierwszy – gracz I (konkurent) wybiera jedną z możliwych alternatyw: przyjąć do realizacji zamówienie (1) lub zamówienia nie przyjmować (2), przy czym istnieją określone prawdopodobieństwa wystąpienia w przetargu konkurentów p_k i p_n.

2. Krok drugi, losowy, w którym jest generowana liczba odpowiadająca przypadkowi zakupienia lub niezakupienia informacji (np. H i R). W przypadku wystąpienia H (i tylko w tym przypadku) gracz II jest informowany o podjętej w pierwszym kroku decyzji gracza I. Praktycznie polega to na tym, że gracz II kupuje od Centrum Gry informację za określoną liczbę jednostek pieniężnych (j.p.). W przeciwnym przypadku informacja nie jest kupowana.

3. Krok trzeci – gracz II wybiera jedną z trzech możliwych strategii określających wartości narzutu: W – wysoka, S – średnia, N – niska.

4. Krok czwarty, losowy – generowanie prawdopodobieństw wyboru wartości narzutu przez gracza I: p_1 – dla wysokiego (W), p_2 – dla średniego (3), p_3 – dla niskiego (N).

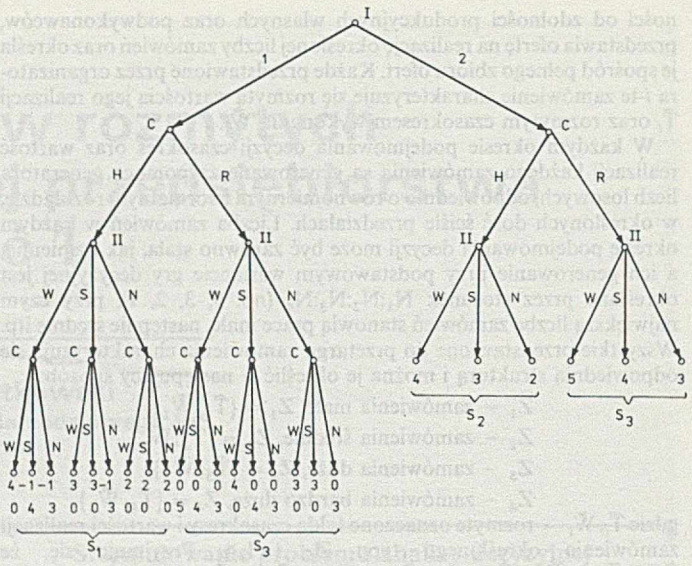
5. Po powyższych krokach kończy się pierwsza partia gry i są realizowane kroki 1–4 dla n partii.

6. Wynik gry. Obliczanie sumarycznych wielkości wygranej dla graczy I i II odbywa się zgodnie ze strategiami przyjętymi na poszczególnych etapach gry.

Dla ilustracji procesu analizy rozpatrywaną grę można przedstawić w postaci drzewa decyzyjnego (dendrytu). Drzewo przykładowej gry przedstawiono na rys. 3. Jego wierzchołkami są różne sytuacje, powstające w procesie gry. Kroki gry przedstawiono strzałkami łączącymi wierzchołki drzewa.

Wierzchołki odpowiadające strategiom graczy oznaczono cyframi rzymskimi I i II, a odpowiadające strategiom losowym – literą C. Końcowe, określające wynik wierzchołki są oznaczone cyframi, określającymi wygraną (przegraną) graczy I i II w czasie jednej partii gry.

Liczba wierzchołków ostatniego, najniższego poziomu dendrytu gry jest równa liczbie możliwych wariantów (realizacji) jednej partii gry.



Rys. 3. Klasy informacji oraz drzewo decyzyjne gry

Dany wariant partii gry określa się konkretnymi realizacjami strategii x i y graczy I i II oraz konkretną realizacją strategii losowej h, tj. $g = (x, y, h)$. Każda gałąź dendrytu gry, od wierzchołka początkowego do jednego z wierzchołków najniższych, odpowiada realizacji jednej z partii gry. Tak np. skrajna lewa gałąź dendrytu przedstawia następujący wariant gry: gracz I w pierwszym kroku przedstawia ofertę na zamówienia (liczba 1), w drugim kroku (strategia losowa) gracz II kupuje informację od Centrum Gry. W trzecim kroku gracz II określa wysoki narzut na realizację danego zamówienia, a w czwartym (strategia losowa) konkurent wybiera wysoki narzut. Wyniki gry dla gracza I wynoszą 0 j.p., natomiast dla gracza II – 4 j.p. (zysk od realizacji zamówienia z wysokim narzutem pomniejszony o jedną j.p. związaną z zakupem informacji). Przy jednakowych narzutach zamówienie otrzymuje gracz, który kupuje informację od Centrum Gry, gdyż ono określiło taki właśnie priorytet.

W rozpatrywanej grze dla gracza II można wydzielić trzy klasy informacji: S_1^{II} – wiadomo, że gracz I wybrał decyzję 1; S_2^{II} – wiadomo, że gracz I wybrał decyzję 2; S_3^{II} – nie wiadomo, jaką decyzję podjął gracz I. Dla gracza I można wydzielić sytuację, w której występuje tylko jedna klasa informacji, a mianowicie S^I, gdy posunięć jeszcze nie było i gracz I wykonuje pierwsze posunięcie, nie znając wyników następnych posunięć.

Rozmyte modele problemów wielokryterialnych

Z problemem podejmowania decyzji zespołowych jest ściśle związany problem wielokryterialnego podejmowania decyzji. Wielokryterialną optymalizację w rozmytym otoczeniu przedstawia się w postaci systemu $\langle X, C_1, ..., C_n, L \rangle$, gdzie X – uniwersalny zbiór alternatyw, L – krata, a kryterium C_i, $i \in 1,n$ przedstawia L – rozmyty zbiór

$$\mu_{C_i} \in F_L(X), \quad F_L(X) = \{\mu_{C_i} \mid \mu_{C_i}: X \to L\}.$$

Rozmyta postać zadania wielokryterialnego wyboru zakłada, że jest znany zbiór porównywalnych alternatyw $A = \{A_1, ..., A_m\}$ i zbiór kryteriów $C = \{C_1, ..., C_n\}$, przy czym rozmyta ocena alternatywy A_j na podstawie kryterium C_i charakteryzuje się funkcją przynależności $\mu_{R_{ij}}(r_{ij}), r_{ij} \in R$, a względna waga w_i kryterium C_i jest funkcją przynależności $\mu_{w_i}(w_i)$, $w_i \in R^+$. Oprócz tego istnieje zbiór optymalnych alternatyw. Szczególnie istotnym problemem w zagadnieniach wielokryterialnego podejmowania decyzji jest wyznaczanie wag w_i. Na ogół w tym przypadku korzysta się z metody opracowanej przez Saaty'ego, opartej na porównywaniu kolejnych par kryteriów.

Problem wielokryterialnej oceny i wyboru, na przykładzie opracowania racjonalnego programu produkcji przedsiębiorstwa, przedstawiono w [18, 20]. Rozpatrzymy teraz wykorzystanie metody Saaty'ego do konstruowania funkcji przynależności zbioru rozmytego „prawdopodobieństwo sukcesu w przetargu" na podstawie jakościowego porównania stopni przynależności ocen poszczególnych decydentów.

Przykład 3

Do określenia prawdopodobieństwa sukcesu w przetargu realizowanego w formie gry decyzyjnej opisanej w [17, 21, 22] określa się możliwość otrzymania zamówień w zależności od wielkości narzutu i sytuacji rynkowej. W rozpatrywanym problemie sytuację rynkową Q przedstawia się w funkcji od m i n tj. $Q = f(m, n)$, gdzie m – liczba przedsiębiorstw uczestniczących w przetargu, n – liczba zamówień przedstawionych do przetargu. Stany określające sytuację rynkową Q są opisane za pomocą terminów lingwistycznych w postaci „bardzo sprzyjająca", „sprzyjająca", „znacznie lepsza od normalnej", „lepsza od normalnej", ..., „bardzo niesprzyjająca". Przy czym np. normalna sytuacja rynkowa występuje wtedy, gdy jest spełniona zależność: $(n\downarrow\downarrow \wedge m\downarrow\downarrow) \vee (n\downarrow \wedge m\downarrow) \vee (n_0 \wedge n_0) \vee (n\uparrow \wedge m\uparrow) \vee (n\uparrow\uparrow \wedge m\uparrow\uparrow)$, gdzie znaki „↑↑", „↑", „0", „↓", „↓↓" oznaczają terminy lingwistyczne zmiennych m oraz n: „bardzo duża", „duża", „średnia", „mała", „bardzo mała".

W wyniku ankietowania decydentów, uwzględniających określone w wyżej przedstawiony sposób sytuacje rynkowe, konstruowana jest macierz $B = \| b_{ii} \|$ o rozmiarze n × n, gdzie n – liczba punktów u_i, w których porównywane są wartości funkcji przynależności. Element b_{ij} macierzy B jest subiektywną oceną relacji $\mu_A(u_i)/\mu_A(u_j)$ i określa, w jakim stopniu w opinii eksperta (decydenta) $\mu_A(u_i)$ jest większe niż $\mu_A(u_j)$. Wielkość b_i jest określana na podstawie skali, na ogół dziewięciopunktowej. Liczba formułowanych pytań do eksperta wynosi $(n^2 - n)/2$, ponieważ $\| b_{ij} \| = 1$. Celem uzyskania zgodności ocen decydenta ustala się, że $b_{ij} = 1/b_{ji}$. Wartość funkcji przynależności $\mu_A(u_1), ..., \mu_A(u_n)$ w punktach $u_1, ..., u_n$ określa się na podstawie znalezionego rozwiązania, przedstawiającego wektor własny macierzy B: $BW^T = \lambda_{max} W$, gdzie λ_{max} – maksymalna wartość własna dla macierzy B; $W = (w_1, ..., w_n)$ – wektor własny spełniający równanie $BW^T = \lambda_{max} W$; T – symbol transpozycji macierzy B.

Dla reprezentacji wiedzy w systemie eksperckim [21, 22] do określenia prawdopodobieństwa sukcesu PROBSU w przetargu wykorzystuje się reguły produkcji typu:

⟨IF narzut rośnie, THEN PROBSU zmniejsza się⟩
⟨IF narzut zmniejsza się, THEN PROBSU rośnie⟩
⟨IF sytuacja rynkowa pogarsza się, THEN narzut zmniejsza się⟩
⟨IF sytuacja rynkowa polepsza się, THEN narzut rośnie⟩ itd.

System ekspercki, realizując proces wnioskowania i wykorzystując wartości prawdopodobieństw warunkowych, zmiennych lingwistycznych zbiorów rozmytych oraz współczynnika pewności, określa prawdopodobieństwa sukcesu w przetargu. Dokładne opisanie zastosowania systemu eksperckiego do określenia prawdopodobieństwa sukcesu w przetargu i wykorzystywanych w nim przedstawionych wyżej zmiennych lingwistycznych przedstawiono w [21, 22, 23]. Jest zrozumiałe, że określeniu „bardzo małe prawdopodobieństwo sukcesu w przetargu" odpowiada określenie „bardzo mały narzut" i odwrotnie przy stałej sytuacji rynkowej.

Lingwistyczne modele podejmowania decyzji

W lingwistycznych modelach podejmowania decyzji posiadana przez decydenta informacja jest przedstawiana za pomocą zmiennych lingwistycznych. Przy określaniu kryteriów stosuje się logikę rozmytą z lingwistycznymi wartościami prawdy, np. przy określaniu preferencji za pomocą zdań typu „x nieznacznie dominuje nad y" itp. Kryterium lingwistycznym nazywamy takie kryterium, którego oceny na skali są wartościami zmiennej lingwistycznej, tj. stanowią rozmyte podzbiory określonego zbioru podstawowego.

Środkiem matematycznego opisu ocen lingwistycznych na podstawie skal mogą być liczby rozmyte. Porównanie wartości lingwistycznych przeprowadza się na podstawie nieostrej relacji porządku, określonego na zbiorze liczb rozmytych. W przypadku, gdy w relacji rozmytej preferencje są określane nie za pomocą liczb, a słownie, mamy do czynienia z lingwistyczną relacją preferencji.

W zależności od charakteru podejmowania decyzji, możliwe jest dwojakie traktowanie lingwistycznych relacji preferencji jako:

- stopnia pewności decydenta, w przypadku gdy ma on wątpliwości i trudności w określeniu swego stanowiska w postaci logiki boolowskiej, np. „priorytetowo" – „niepriorytetowo"
- intensywności jednoznacznej, dokładnej preferencji decydenta.

W [16] rozpatrzono wykorzystanie teorii zbiorów rozmytych do konstruowania funkcji przynależności „uszeregowanie obiektów" przy opracowaniu harmonogramów pracy maszyn. Obecnie rozpatrzymy określenie lingwistycznej relacji preferencji przy analizie „okien" czasowych, tj. przedziałów czasu, w ciągu których powinny być wykonane określone zamówienia [3]. Na podstawie względnego rozmieszczenia przedziałów czasowych $[t_k, F_k]$ oraz $[t_l, F_l]$ charakteryzujących zadanie k i l, a także pracochłonności tych zadań, można określić ograniczenie dotyczące kolejności ich wykonania lub stwierdzić niemożliwość ich realizacji, gdy istniejące „okna" czasowe są bardzo wąskie. Ma to istotne znaczenie przy określaniu rzeczywistej informacji o możliwych opóźnieniach wykonania zamówień w procesie planowania produkcji.

Przykład 4

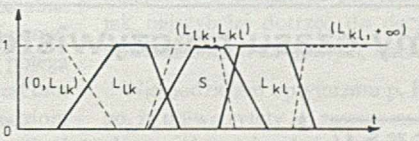
Załóżmy, że przy wykonywaniu każdego zadania (zamówienia) trzeba wykorzystać zasoby określonego typu. Niech M – zbiór maszyn, a m_k – konkretna maszyna przeznaczona do wykonania zadania k. Zakłada się, że zadanie jest wykonywane bez przerwy i że na jednej maszynie można w tym samym czasie wykonywać tylko jedno zadanie. Niech k i l to takie dwa zadania, że $m_k = m_l$ oraz, że nie mogą być one wykonywane jednocześnie, nawet jeżeli nie istnieją a priori ograniczenia dotyczące kolejności ich wykonania. Przy czym t_k – najwcześniejszy, a I_k – najpóźniejszy moment rozpoczęcia k-go zadania, uwarunkowane możliwościami opóźnień, przy czym d_k – pracochłonność wykonania k-go zadania. W ten sposób zadanie k jest scharakteryzowane za pomocą „okna" czasowego $I_k = [t_k, F_k]$, gdzie $F_k = T_k + d_k$. Na podstawie względnego rozmieszczenia odcinków czasowych $[t_k, F_k]$ oraz $[t_l, F_l]$ rozpatruje się możliwość ich wykonania.

Analizę tę przeprowadza się w następujący sposób:

- Jeżeli $d_k + d_l > \max(F_k - t_l, F_l - t_k)$, to „okno" czasowe $I_k \cup I_l$ jest zbyt wąskie do wykonania dwóch zadań. W tym przypadku należy rozpatrzeć ograniczenia dotyczące przemieszczenia w czasie wykonania tych zadań;
- Jeżeli $\min(F_k - t_l, F_l - t_k) < d_k + d_l \leqslant \max(F_k - t_l, F_l - t_k)$, to wprowadza się ograniczenie kolejności wykonania zadań k i l.
  Przy $F_k - t_l \geqslant F_l - t_k$, zadanie l powinno być wykonane przed zadaniem k; $I_k \cap I_l \neq 0$ i odcinek I_l całkowicie wyprzedza odcinek I_k;
- Jeżeli $d_k + d_l \leqslant \min(F_k - t_l, F_l - t_k)$, to zadania mogą być wykonywane w dowolnej kolejności. Jednakże przy $F_k - t_l > F_l - t_k$ wykonanie tych zadań w kolejności początkowo l, a następnie k pozwala zachować dużą rezerwę czasu.

Po określeniu priorytetu wykonania zadania l przed zadaniem k można zmniejszyć szerokość „okien", wtedy odcinek I_k ma postać: $[\max(t_k, t_l + d_l), F_k]$, a odcinek I_l – postać: $[t_l, \min(F_l, F_k - d_k)]$. Te warunki także uogólnia się na przypadek, gdy najwcześniejsze terminy rozpoczęcia prac t_k, t_l, wielkości F_k, F_l oraz d_k i d_l są wartościami rozmytymi.

Rozmieszczenie wielkości rozmytej S, w stosunku do dwóch rozmytych odcinków L_{kl} oraz L_{lk} przedstawiono na rys. 4.



Rys. 4. Rozmieszczenie rozmytej liczby S w stosunku do dwóch rozmytych odcinków L_{kl} i L_{lk}

Położenie punktu ze współrzędnymi μ_{kl}, μ_{lk} w kwadracie jednostkowym

μ_{lk}

Zadanie k powinno być wykonane przed zadaniem l	Zadanie k ma priorytet przed zadaniem l	Relacja obojętności między priorytetem wykonywania zadań k oraz l
Zadanie k powinno być wykonywane przed zadaniem l /ryzyko sprzeczności/	Sytuacja nieokreślona	Zadanie l ma priorytet przed zadaniem k
Całkowita sprzeczność	Zadanie l powinno być wykonywane przed zadaniem k /ryzyko sprzeczności/	Zadanie l powinno być wykonywane przed zadaniem k

0 μ_{kl}

Macierz lingwistycznej relacji preferencji na zbiorze możliwych uporządkowań wykonania dwóch zamówień k i l może być przedstawiona za pomocą kwadratu jednostkowego, określonego przez funkcje przynależności μ_{lk} i μ_{kl} (p. tabela), gdzie μ_{lk} i μ_{kl} określają stopnie, na ile wielkość rozmyta S jest mniejsza lub równa L_{lk} lub L_{kl}.

Szczególnie sporna sytuacja może powstać przy analizie rozmieszczenia punktu w kwadracie jednostkowym ze współrzędnymi $\mu_{lk} = \mu_{kl} = 0{,}5$, gdyż wtedy liczby rozmyte L_{kl} i L_{lk} nakładają się wzajemnie, a wielkość S znajduje się wewnątrz nośnika ich wspólnej części. Wtedy nic nie można przewidzieć ze względu na niedokładności wartości opóźnień.

Modele rozmytego programowania dynamicznego oraz inne metody reprezentacji i pozyskiwania wiedzy ilustrowane na wyżej przedstawionym problemie decyzyjnym zostaną rozpatrzone w drugiej części artykułu.

LITERATURA

[1] Bloos J.: Marketing praktyczny. Buyar Marketing, Warszawa 1992
[2] Bolc L., Borodziewicz W., Wójcik M.: Podstawy przetwarzania informacji niepewnej i niepełnej. PWN, Warszawa, 1991
[3] Dubois D., Prade H.: Teoria możliwości. Zastosowanie do reprezentacji wiedzy w informatyce (przekł. z franc.). Radio i Swjaz, Moskwa 1990
[4] Gawriłowa T.A., Czerwińskaja K.R.: Izwljeczjenie i strukturirowanije znanij dla ekspertnych sistjem. Radio i Swjaz, Moskwa 1992
[5] Inmon W.H.: Information Engineering For The Practitioner. Putting Theory Into Practice. Prentice Hall, Englewood Cliffs 1988
[6] Kacprzyk J.: Zbiory rozmyte w analizie systemowej. PWN, Warszawa 1986
[7] Klir G.J., Folger T.A.: Fuzzy sets, uncertainty and information. Prentice Hall, Englewood Cliffs 1988
[8] Kofler E.: Podejmowanie decyzji przy niepełnej informacji. Real Publishers, Warszawa 1993
[9] Krawczyk S.: Matematyczna analiza sytuacji decyzyjnych, PWE, Warszawa 1990
[10] Nieczjotkie mnożjestwa w modeliach uprawljenija i iskusstwjennogo intjelljekta. Pod red. D.A. Pospiełowa, Nauka, Moskwa 1986
[11] Ostasiewicz W.: Zastosowanie zbiorów rozmytych w ekonomii. PWE, Warszawa 1986
[12] Obrabotka niecziotkoj informacji w sistiemach priniatja rieszenij. Pod red. A.W. Borisowa. Radio i Swiaz. Moskwa 1990
[13] Pozyskiwanie wiedzy (przekł. z jap.) Mir, Moskwa 1990
[14] Przetwarzanie wiedzy (przekł. z jap.) Mir, Moskwa 1989
[15] Reprezentacja i wykorzystanie wiedzy (przekł. z jap.) Mir, Moskwa 1989
[16] Witkowski T.: Algorytm rozpoznawania problemu szeregowania zadań. Informatyka nr 6, 1993
[17] Witkowski T.: Fuzzy Logic in Decision Making During Simulation of Enterprises Activities under Circumstances of Tenders. Abstracts of the IFORS 2nd Specjalized Conference on „Transition to Advanced Market Economies", Warsaw June 22–25, 1992. System Research Institute, Polish Academy of Sciences, Warsaw 1992

Systemy czasu rzeczywistego

dokończenie ze s. 17

[10] MacEven G.H., Skilicorn D.B.: Using Higher-order Logic for Modular Specification of Real-Time Distributed Systems. W: Joseph (ed.): Formal Techniques in Real-Time and Fault-Tolerant Systems. LNCS, Vol. 331, pp. 36–66, Springer-Verlag, 1988
[11] Maggiolo-Schettini A., Winkowski J.: Towards an Algebra for Timed Behaviours. Prace IPI PAN, 677, 1990
[12] Merlin P., Farber D.J.: Recoverability of Communication Protocols – Implications of Theoretical Study. IEEE Trans. Comm., Vol. COM-24, pp. 1036–1043, 1976
[13] Peng D., Shin K.G.: Modeling of Concurrent Task Execution in a Distributed Systems for Real-Time Control. IEEE Trans. on Computers, Vol. C-36, pp. 500–516, 1987
[14] Peterson J.L.: Petri Net Theory and the Modelling of Systems. Prentice-Hall, 1981
[15] Pnueli A., Harel E.: Application of Temporal Logic to the Specification of Real-Time Systems. W: Joseph (ed): Formal Techniques in Real-Time and Fault-Tolerant Systems. LNCS, Vol. 331, pp. 84–98, Springer-Verlag, 1988
[16] Ramchandani C.: Analysis of Asynchronous Concurrent Systems by Petri Nets. MIT, Cambridge, Project MAC, Rep. TR-120, 1974
[17] Tsai J. J-P, Weigert T.J.: Knowledge-Based Software Development for Real-Time Distributed Systems. World Scientific, 1993.
[18] Witkowski T.: Metoda analizy hierarchicznej w projektowaniu i zarządzaniu systemem produkcyjnym. Przegląd Mechaniczny nr 20, 1993
[19] Witkowski T.: Metodologia wykorzystania rozmytych zdań warunkowych przy podejmowaniu decyzji w zarządzaniu produkcją. Kibiernietika i wyczislitielnaja technika, nr 99, 1993
[20] Witkowski T.: Racjonalny wybór programu produkcyjnego przedsiębiorstwa. Przegląd Organizacji nr 12, 1993
[21] Witkowski T.: The characteristic function and expert system constructions determining probable success in the competition tenders. Preprints of the IFAC/IFORS/IIASA/TIMS Workshop on „Supprot Systems for Decision and Negotiation Process", Warsaw June 24–26. Systems Research Institute Polish Academy of Sciences, Warsaw 1992
[22] Witkowski T.: Zastosowanie systemu eksperckiego w określaniu prawdopodobieństwa sukcesu w przetargu. Przegląd Organizacji nr 12, 1992
[23] Witkowski T.: Pozyskiwanie wiedzy i konstrukcja reguł wnioskowania dla oceny prawdopodobieństwa sukcesu w przetargu. Materiały krajowej konferencji „Analiza decyzyjna. Systemy eksperckie. Zastosowanie systemów komputerowych". Warszawa, 25–27 maja 1994 r. Instytut Badań Systemowych PAN, Warszawa 1994.

Nowe książki

AKADEMICKA OFICYNA WYDAWNICZA PLJ

Kazimierz Subieta: Ingres: – system zarządzania relacyjną bazą danych. Warszawa 1994 r., wyd. 1, s. 132. ISBN 83-7101-144-X

W porównaniu z innymi podobnymi produktami, system zarządzania relacyjną bazą danych Ingres zajmuje szczególną pozycję inowacyjnego lidera, stale wprowadzającego nowe jakości do swoich mechanizmów i interfejsów. Ingres jako pierwszy wprowadził język 4GL do programowania relacyjnych baz danych, pierwszy zbudował bramki do obcych systemów, pierwszy zrealizował przezroczystą rozproszoną bazę danych, pierwszy wprowadził elementy „inteligencji" do bazy danych i pierwszy zrealizował język 4GL oparty na komponowaniu interfejsu użytkownika z elementów graficznych. Są to powody, dla których Ingres spotyka się ze szczególnym zainteresowaniem zarówno ze strony jego potencjalnych użytkowników, jak i świata naukowego.

Wychodząc naprzeciw temu zainteresowaniu, książka przedstawia podstawowe cechy systemu Ingres, szczególnie różne aspekty jego architektury i własności funkcjonalnych. Szczególną uwagę poświęca się narzędziom, umożliwiającym tworzenie zastosowań z bazą danych, w tym szerokiej gamie języków programowania. Ich ważną odmianą są języki czwartej generacji (4GL). Omówione są różnorodne zastosowania języka SQL, stanowiącego w systemie Ingres podstawowy środek dostępu do bazy danych. Przedstawione są cechy systemu uzasadniające użycie określenia „inteligentny", w tym aktywne reguły i zarządzanie obiektami. Wyjaśnione są założenia architektury klient-serwer, dostęp do odległych baz danych oraz bramki umożliwiające dostęp do obcych baz danych. Prezentacja jest uzupełniona wieloma przykładami.

Biorąc pod uwagę trudności, jakie może napotkać Czytelnik przy zetknięciu z „bazo-danowym" kosmosem, istotnym fragmentem książki jest słownik, wyjaśniający w przystępny sposób znaczenie podstawowych terminów i akronimów występujących w literaturze poświęconej bazom danych.

Ingres stanowi kwintensencję wielu koncepcji, teorii i praktycznych rozwiązań w dziedzinie baz danych. Materiał zawarty w książce powinien zainteresować wielu informatyków chcących zdobyć orientację ogólną w zakresie baz danych, jak i szczegółową w zakresie własności oferowanych przez system Ingres. Książka może stanowić podstawę wstępnego kursu systemu (6-16 godzin) na wydziałach informatycznych.

Akademicka Oficyna Wydawnicza PLJ: 02-323 Warszawa, ul. Radomska 16 m. 2, tel. 22-20-01

Odpowiedź na polemikę „Co to jest uniwersytet?" Andrzeja Horodeńskiego, zamieszczoną w numerze 6/94 ENTER

Z wielką uwagą przeczytałem w czerwcowym numerze ENTER polemikę „Co to jest uniwersytet? pióra Andrzeja Horodeńskiego, stanowiącą – jak dotąd – jedyną odpowiedź na mój artykuł „Logika teoretycznych urojeń" zamieszczony w czasopiśmie INFORMATYKA z grudnia ub.r. Polemika jet nietypowa: w większości p. Horodeński solidaryzuje się z moimi poglądami. Podzielam pogląd p. Horodeńskiego na temat przyczyn milczenia teoretyków, tak niegdyś pewnych jedynej słuszności metod matematycznych w informatyce. Końcówka polemiki zawiera jednak tezy, które chciałbym skomentować.

Zacznę od siebie. Pan Horodeński sugeruje, że jako praktyk zaatakowałem uczonych za to, że nie robią praktyki – czyli jako osoba z zewnątrz próbuję dyktować autonomicznym uniwersytetom, co mają robić. Wyjaśniam, że pochodzę ze środka nauki akademickiej i działam w teorii, w praktyce i w dydaktyce, zarówno w Polsce jak i za granicą. Ponieważ powierzono mi organizowanie zespołu baz danych i inżynierii oprogramowania w Instytucie Podstaw Informatyki PAN, sprawą zasadniczej wagi stało się dla mnie usunięcie z przedpola mojej działalności pozorantów, formalistycznie łączących zmatematyzowane teorie z modnymi lub ważnymi terminami informatycznymi, dla uzyskania niezasłużonej przewagi w ocenach różnych ciał naukowych. Książka G. Mirkowskiej i A. Salwickiego, nominalnie dotycząca inżynierii oprogramowania, stanowiła dla mnie drastyczny przykład tej postawy.

Główna teza polemiki p. Horodeńskiego głosi, że nie jest rolą uniwersytetów robienie systemów użytkowych; od tego jest przemysł. Podstawą argumentacji jest rozumowanie: jeżeli nie (zmatematyzowana) teoria, to praktyka, czyli systemy użytkowe. Ta wygodna dla teoretyków informatyki fałszywa dychotomia nia pozostawia miejsca dla innych metod naukowych. Jak piszę w moim artykule, **większość**, odkryć naukowych w dziedzinie informatyki jest skutkiem eksperymentów polegających na budowie prototypowych (niekoniecznie komercyjnych) systemów. Budowa eksperymentalnego systemu kształci dziesiątki studentów, magistrantów i doktorantów, tworzy i rozwiązuje problemy, stwarza podstawy do klasyfikacji, syntez i uogólnień. Nie można zajmować się dużymi systemami informatycznymi bez wniknięcia w ich anatomię projektową i implementacyjną, tak samo jak nie można nauczyć się grać na skrzypcach bez wzięcia do ręki instrumentu, lub pływać bez wejścia do wody. Wbrew temu, co twierdzi pan Horodeński, na dobrych wydziałach informatycznych zachodnich uczelni badania teoretyczne są marginesem. Podstawowa działalność polega na realizacji niezliczonej liczby eksperymentalnych projektów i do rzadkości należy osiągnięcie przez nie statusu komercyjnego (sam uczestniczyłem w kilku takich projektach).

Zły klimat dla tego rodzaju badań eksperymentalnych na polskich uczelniach powoduje, że trudno tam znaleźć pracowników naukowo-dydaktycznych, którzy potrafią studentom wyjaśnić „ ...jak powinni robić Oracle'a, Ingresa i WordPerfecta...", natomiast stosunkowo łatwo znaleźć specjalistów od logiki, dedukcyjnych baz danych, sztucznej inteligencji i innych teoretycznych marginalii. Te marginalia na ogół mają znikome szanse stania się tym, co w języku angielskim określa się terminem *technology* – technologia. Zadaniem kształcenia jest przede wszystkim wyposażenie absolwentów studiów w wiedzę dotyczącą aktualnych i potencjalnych technologii. Zadaniem informatyki jako nauki – rozwój i tworzenie nowych technologii. Uczelnie mogą mieć lub mogą nie mieć teoretyków informatyki, natomiast **muszą** dysponować specjalistami z kluczowych dziedzin technologicznych. Wymieniam dla porządku: języki i systemy programowania, bazy danych, systemy operacyjne, architektura komputerów, sieci komputerowe, projektowanie systemów informatycznych, inżynieria oprogramowania, systemy rozproszonego przetwarzania, grafika komputerowa, zastosowania multi-medialne. Są to dziedziny naukowo, intelektualnie i materialnie bardzo atrakcyjne. Problem polega jednak na tym, że bariera kompetencji jest tam podniesiona na wysoki poziom i wielu ludziom brak jest wytrwałości do jej pokonania. Stąd tendencje do pozorowania kompetencji, lub ucieczki w tematykę marginalną i egzotyczną, typu „zbiory przybliżone".

W zakresie wymienionych dziedzin technologicznych stan na polskich uczelniach jest dramatyczny, owocujący degradacją rangi studiów informatycznych w Polsce i stopniowo sprowadzający nasz kraj do roli trzeciego świata, czyli rynku zbytu drogich i przestarzałych produktów zachodnich. Głównymi przyczynami są bieda, emigracje naukowców i ogromne zapotrzebowanie zewnętrzne na specjalistów znających się na tych dziedzinach, ale istotny udział ma tu także poddawanie się środowisk akademickich presji utytułowanych teoretyków informatyki, tworzących zwarte mafijne krążki wzajemnego wspierania swoich interesów i uzurpujące sobie monopol na „naukowość". Brak zdecydowanej polityki naukowej sprzyja takim patologiom. Do nich zaliczam nie tylko pseudoteorie, ale również rozpanoszenie się tematyki „sztucznej inteligencji" i „programowania w logice".

Czy spór „praktycznie czy teoretycznie"jest na uniwersytetach stale weryfikowany na korzyść teorii? Moim zdaniem, takiego sporu nie ma. Pan Horodeński idealistycznie utożsamia „czystą naukę" z budowaniem teorii, przeciwstawiając je praktyce. To przeciwstawienie jest sprzeczne w samym założeniu. W naukach dotyczących technologii teoria i praktyka są nierozłączne jak dwie strony tej samej monety. Spór może dotyczyć wyłącznie metod uprawiania nauki, a metody mogą być różne: teoretyczna synteza, budowa modeli matematycznych, spekulacje filozoficzno-ideologiczne, opis i klasyfikacja zjawisk i obiektów rzeczywistych, badania empiryczne, symulacja, eksperyment, i inne. Celem mojego artykułu było napiętnowanie bezpłodnej metody naukowej polegającej na budowie nigdy nie weryfikowanych zmatematyzowanych ideologii w dziedzinie informatyki (niesłusznie zwanych „teoriami" – ponieważ brakuje im drugiej strony wspomnianej wyżej monety). Jestem zdania, że ta metoda jest martwa i fakt ten już dotarł do powszechnej świadomości informatyków. Uważam, że powinien on również jak najszybciej dotrzeć do decydentów odpowiedzialnych za politykę naukową.

Nie podzielam optymizmu p. Horodeńskiego, że uniwersytety „...zawsze znajdowały siłę, aby wyjść z pułapek intelektualnych...". Bez krytyki i aktywnego przeciwstawienia się silnej grupie partykularnego interesu, którą tworzą utytułowani teoretycy informatyki, nic w cudowny sposób samo się nie zreperuje. Uczelnie i inne instytucje naukowe muszą podjąć zdecydowaną walkę o właściwy kształt nauk informatycznych, a mój artykuł i polemika p. Horodeńskiego stanowią być może jej początkowe elementy.

KAZIMIERZ SUBIETA

Przypominamy!

Informatyki nie ma w sprzedaży kioskowej, istnieje tylko w prenumeracie. W każdej chwili można ją zaprenumerować na dowolny czas. Wystarczy zadzwonić do redakcji (Warszawa, tel.: 39-14-34), udzielimy wyczerpujących informacji, jak to należy uczynić. Można także u nas kupić wszystkie numery archiwalne.

Żurakowski Z.: Niektóre zagadnienia związane z projektowaniem i eksploatacją oprogramowania na poziomie profesjonalnym
INFORMATYKA 1994, nr 8, s. 4
Ocena aktualnej krajowej sytuacji oraz postulaty w zakresie rozwoju systemów czasu rzeczywistego w Polsce.

Żurakowski Z.: Some problems connected with design and operation of software on professional level
INFORMATYKA 1994, No. 8, p. 4
Evaluation of the actual situation in Poland and requirements for development of real-time systems.

Żurakowski Z.: Einige mit Softwareprojektierung und -einsatz auf professionellem Niveau verbundene Probleme
INFORMATYKA 1994, Nr. 8, S. 4
Beurteilung heutiger Lage und Forderungen im Bereich der Entwicklung von Echtzeitsystemen in Polen.

Górski J., Wardziński A.: Analiza bezpieczeństwa komputerowego systemu sterowania
INFORMATYKA 1994, nr 8, s. 7
Charakterystyka dwóch metod analizy bezpieczeństwa działania komputerowego systemu sterowania na przykładzie zastosowania systemu medycznego Therac-25.

Górski J., Wardziński A.: Analysis of computer control system security
INFORMATYKA 1994, No. 8, p. 7
Characteristics of the two analysis methods for computer control system operation security on example of the Therac-25 medical system.

Górski J., Wardziński A.: Analyse der Sicherheit von Computersteuersystem
INFORMATYKA 1994, Nr. 8, S. 7
Eine Charakteristik von zwei Sicherheitsanalysemethoden für Computersteuersystem am Beispiel von Anwendung des Therac-25-Medizinsystems.

Huzar Z.: Projektowanie oprogramowania systemów czasu rzeczywistego. Podstawowe fazy projektu
INFORMATYKA 1994, nr 8, s. 11
Charakterystyka abstrakcyjnego modelu czasu rzeczywistego oraz podstawowych pojęć związanych ze specyfikacją i implementacją tego typu systemu.

Huzar Z.: Design of software for real-time system. Basic design phases
INFORMATYKA 1994, No. 8, p. 11
Characteristics of an abstract real-time system model and of basic terms connected with specification and implementation of such systems.

Huzar Z.: Projektierung von Software der Echzeitsysteme. Grundphasen eines Projektes
INFORMATYKA 1994, Nr. 8, S. 11
Eine Charakteristik des abstrakten Modells von einem Echtzeitsystem und der Grundbegriffe, die mit Spezifikation und Implementierung eines solchen Systems verbunden sind.

Huzar Z., Magott J.: Systemy czasu rzeczywistego – metody specyfikacji
INFORMATYKA 1994, nr 8, s. 14
Charakterystyka wybranych metod specyfikacji systemów czasu rzeczywistego.

Huzar Z., Magott J.: Real-time systems – specification methods
INFORMATYKA 1994, No. 8, p. 14
Characteristics of selected real-time system specification methods.

Huzar Z., Magott J.: Echtzeitsysteme – Spezifikationsmethoden
INFORMATYKA 1994, Nr. 8, S. 14
Eine Charakteristik der ausgewählten Spezifikationsmethoden von Echzeitsystemen.

Sacha K.: Standardy planowania jakości, dokumentacji i konserwacji oprogramowania
INFORMATYKA 1994, nr 8, s. 18
Omówienie istniejących standardów ANSI/IEEE dotyczących planowania i kontroli jakości oprogramowania, dokumentowania projektów programowych oraz organizacji procesów konserwacji i atestacji zmian.

Sacha K.: Standards for software quality planning, documentation and maintenance
INFORMATYKA 1994, No. 8, p. 18
Discussion of existing ANSI/IEEE standards concerning software planning an quality checking, documentation and organization of maintenance, as well as certification of changes.

Sacha K.: Standards für Softwarequalitätsplanung, Dokumentation und Pflege
INFORMATYKA 1994, Nr. 8, S. 18
Eine Besprechung von bestehenden ANSI/IEEE-Standards betreffend Softwarequalitätsplanung und -kontrolle, Dokumentation der Softwareprojekte, sowie Organisation der Pflege- und Änderungenattestierungsprozesse.

Hudyma E.: Testowanie oprogramowania
INFORMATYKA 1994, nr 8, s. 22
Charakterystyka celów, kryteriów i podstawowych pojęć testowania oprogramowania, a także metod stosowanych w poszczególnych fazach tworzenia oprogramowania.

Hudyma E.: Software testing
INFORMATYKA 1994, No. 8, p. 22
Characteristics of software testing aims, criteria and basic terms, as well as of methods applied in single phases of software creating.

Hudyma E.: Softwaretestung
INFORMATYKA 1994, Nr. 8, S. 22
Eine Charakteristik von Zielen, Kriterien und Grundbegriffen des Softwaretestens, sowie von Verfahren, die in einzelnen Phasen des Softwareherstellungsprozesses verwendet werden.

Witkowski T.: Metody zbiorów rozmytych w analizie decyzyjnej przedsiębiorstwa. Część 1
INFORMATYKA 1994, nr 8, s. 26
Pierwsza część przeglądu metod i modeli zbiorów rozmytych, stosowanych w procesach decyzyjnych przedsiębiorstwa.

Witkowski T.: Fuzzy sets method in enterprise's decision analysis. Part 1
INFORMATYKA 1994, No. , p. 26
First part of a preview of fuzzy sets methods and models applied in enterprise's decision process.

Witkowski T.: Fuzzy Sets-Methoden in Unternehmensbeschlussanalyse. Teil 1
INFORMATYKA 1994, Nr. 8, S. 26
Erster Teil einer Übersicht von Fuzzy Sets-Methoden und Modellen, die in Unternehmensbeschlussprozess angewendet werden.

Z UNIX-a do Windows NT

Firma DataFocus Inc., partner Microsoft, w ramach programu Microsoft Solution Provider oferuje pakiet programów NuTCRACKER (Dziadek do orzechów), przeznaczonych do przenoszenia aplikacji z UNIXa do środowiska API Win32 systemu Windows NT. Rekompilacja kodu źródłowego aplikacji z UNIXa i połączenie z biblioteką dynamiczną DLL programu NuTCRACKER tworzy 32-bitowy program, zgodny z Win32 API. Pakiet zawiera zbiór najczęściej używanych odwołań systemu UNIX wersje SVR4 i POSIX. Programy NuTCRACKER współpracują z kompilatorami C i C++.

Wielowarstwowe dyski optyczne IBM

Dziesięć lat pracowali naukowcy z Centrum Badawczego IBM w Almaden (San Jose w Kalifornii) nad wielowarstwowymi dyskami optycznymi. Dyski te uzyskuje się przez zestawienie dwu lub więcej warstw dyskowych. Ruchoma głowica lasera wybiera do odczytu jedną z warstw. Składający się z 10 warstw dysk pozwoli na zapis do 6,5 GB informacji. Wielowarstwowe dyski pozwolą m.in. na zapisanie w postaci cyfrowej typowego filmu komercyjnego czy też na zwiększenie pojemności wielodyskowych bibliotek optycznych, opartych na wykorzystaniu zmieniacza dysków („jukebox") (czas dostępu rzędu 10 s). Technologia opracowana przez naukowców z IBM polega na złożeniu pojedynczych dysków w stos, z takimi odstępami między dyskami, aby mogła wejść między nie głowica. Dane zapisuje się na dowolnej wewnętrznej powierzchni dysków. Maksymalna liczba dysków w zestawie jest ograniczona przez moc lasera, przezroczystość warstw oraz koszt wykonania konstrukcji wielowarstwowej. Obecnie można bezbłędnie odczytywać z zestawów składających się z 2, 4 i 6 dysków. W przyszłości wielowarstwowe dyski optyczne – jak stwierdził M. Engler, dyrektor programu w IBM Optical Storage Laboratory – osiągną pojemność 30 GB lub więcej.

Przenośna stacja robocza Suna

SPARCStation Voyager – to przenośna stacja robocza Suna (SMCC). Wykorzystano w niej po raz pierwszy jednostkę centralną SPARC II, drugi chip z serii microSPARC zaprojektowany wspólnie przez SMCC i Fujitsu. Procesor ten jest taktowany zegarem 60 MHz, zaś cała stacja robocza ma dwukrotnie lepsze osiągi od systemów Suna opartych na procesorze microSPARC pierwszej generacji. Voyager jest standardowo wyposażony w wyświetlacz ciekłokrystaliczny z aktywną matrycą, 16 MB RAM, 2,5 calowy twardy dysk o pojemności 340 MB, 3,5 calowy napęd dyskietek o potrójnej gęstości zapisu, standardowe interfejsy wejścia-wyjścia, m.in. 16-bitową kartę dźwiękową stereo, Ethernet, ISDN, SCSI-2, dwa sloty PCMCIA. Voyager jest pierwszą stacją roboczą z interfejsem podczerwieni kompatybilnym z obydwoma czołowymi standardami – interfejsem SIR Hewlett-Packarda i interfejsem Sharp/Apple. Urządzenie to umożliwia bezprzewodową wymianę danych między systemami Suna, a osobistymi asystentami.

SPARCStation Voyager może komunikować się z siecią klient-serwer za pomocą prostego urządzenia wykorzystującego modem i linię telefoniczną. SPARCStation Voyager zajmuje zaledwie 35% powierzchni wymaganej przez tradycyjne stacje robocze SPARCStation i waży ok. 6 kg. Cena standardowej stacji z wyświetlaczem monochromatycznym wynosi 12 295, z wyświetlaczem kolorowym – 13 445 USD.

PIERWSZY POLSKI INTELIGENTNY
SYSTEM INFORMATYCZNY WSPOMAGAJĄCY ZARZĄDZANIE
PRODUKCJĄ - (TPPiRP), FINANSAMI - KOSZTAMI, SPRZEDAŻĄ, KADRAMI
WRAZ Z SYSTEMEM INFORMACJI KIEROWNICTWA
WSPOMAGANY SYSTEMEM SZTUCZNEJ INTELIGENCJI PC-SHELL
ORAZ EKSPERCKĄ BAZĄ WIEDZY EKONOMICZNO-FINANSOWEJ - ISAF
(DLA DUŻYCH PRZEDSIĘBIORSTW PRODUKCYJNYCH)

Perfect-Ekspert AITECH

(DZIAŁAJĄCY W OPARCIU O ŚRODOWISKO PRACY - WYŁĄCZNIE SYSTEM OPERACYJNY DOS, NetWare v3.11, v4.01 FIRMY NOVELL)

OFERUJE

PRZEDSIĘBIORSTWO PRODUKCYJNO-USŁUGOWE INNOWACJI TECHNICZNYCH

JUNISOFTEX Sp. z o.o.

ROK ZAŁOŻENIA 1987
WARTOŚĆ SPRZEDAŻY wg F-02 1992 (14,6 mld), wg F-01 na 1993.10.31 (15,4 mld)
TEL/FAX 315-131, 315-700, 321-525, 388-052, 388-054 TLX 036233
BIURO TECHNICZNO-HANDLOWE
44-100 GLIWICE ul. Kościuszki 1
WSZYSTKICH ZAINTERESOWANYCH ZAPRASZAMY NA MERYTORYCZNE
ROZMOWY DO NASZEGO BIURA TECHNICZNO-HANDLOWEGO
OFERUJEMY RÓWNIEŻ INTELIGENTNE NARZĘDZIE DO PRODUKCJI EKSPERCKICH BAZ WIEDZY

PC-SHELL

ARTIFICIAL INTELLIGENCE

DLA JEDNOSTEK BUDŻETOWYCH (NAUKA, OŚWIATA, URZĘDY MIAST I GMIN, SŁUŻBA ZDROWIA) BEZ PRAWA DO KOMERCYJNEGO WYKORZYSTYWANIA W CENIE 5.000.000,- (BEZ VAT) - INFORMACJE I ZAMÓWIENIA PROSIMY SKŁADAĆ BEZPOŚREDNIO U PRODUCENTA ARTIFICIAL INTELLIGENCE LABORATORY - AITECH, KATOWICE 14 SKR. POCZT. 2804.
DLA POZOSTAŁYCH JEDNOSTEK GOSPODARCZYCH BEZ PRAWA KOMERCYJNEGO WYKORZYSTYWANIA TJ. WYŁĄCZNIE NA WŁASNE POTRZEBY DO KOŃCA I KW. 1994 ROKU W CENIE PROMOCYJNEJ 10.000.000,- (BEZ VAT), DLA KUPUJĄCYCH DO CELÓW KOMERCYJNYCH DO KOŃCA I KW. 1994 25.000.000,- (BEZ VAT), LICENCJA OGRANICZONA NA NIEOKREŚLONĄ ILOŚĆ ODBIORCÓW KOŃCOWYCH.
POSZUKUJEMY DO WSPÓŁPRACY LUB NAWIĄŻEMY WSPÓŁPRACĘ Z EKSPERTAMI Z DOWOLNYCH DZIEDZIN NAUKI I TECHNIKI.

0/3/93

MIĘDZYNARODOWE TARGI KATOWICKIE
ZAPRASZAJĄ NA

SOFT '94 ARG 13-16.09

VIII MIĘDZYNARODOWE TARGI OPROGRAMOWANIA SOFTARG'94

13–16 września 1994 r.
godz. 10–17

Organizowane pod patronatem
Polskiego Towarzystwa Informatycznego

Zakres tematyczny Targów obejmuje następujące obszary zastosowań:

- ★ badania naukowe i pomiary
- ★ dydaktykę
- ★ gromadzenie i wyszukiwanie informacji
- ★ oprogramowanie systemowe i narzędziowe
- ★ sterowanie procesem produkcyjnym
- ★ systemy biurowe, rozliczeniowe i bankowe
- ★ wspomaganie projektowania
- ★ zarządzanie
- ★ transmisja danych

Targom towarzyszą pokazy własne firm oraz konkurs na najlepszy produkt programowy.

MIĘDZYNARODOWE TARGI KATOWICKIE
40 955 KATOWICE
UL. BYTKOWSKA 1B
TEL. (032) 59 60 61...7, (032) 59 83 12
FAX. (03) 15 40 227, (032) 588 919

ZAPRASZAMY!!! DOJAZD Z CENTRUM KATOWIC AUTOBUSAMI: 0, 30, 110, 190

327/94